Altos UNIX® System V/386 Release 3.2

User's Reference (C, M, F)

Document History

EDITION	PART NUMBER	DATE
Preliminary Edition	690-23414-001A	February 1990
First Edition	690-23414-001	April 1990
Second Edition	690-23414-002	March 1991

Copyright Notice

Trademarks

Limitations

GUIDE TO YOUR ALTOS UNIX® SYSTEM V/386 RELEASE 3.2 DOCUMENTATION

RUN-TIME SYSTEM

These books come with every system:

Installation Guide
Part Number: 690–24096–*nnn*
- Operating System installation
  Upgrade procedure

System Administrator's Guide
Part Number: 690–23415–*nnn*
- Sysadmsh
- Security
- System tuning, troubleshooting
- Peripherals
- Virtual Disks

User's Guide
Part Number: 690–23408–*nnn*
- Vi, ed, mail, awk, sed
- Shells: sh and csh
- Job scheduling commands

User's Reference (C, M, F)
Part Number: 690–23414–*nnn*
(also provided online with each operating system)
- (C) Commands
- (M) Miscellaneous files and commands
- (F) File formats

System Administrator's Reference (ADM, HW)
Part Number: 690–23416–*nnn*
(also provided online with each operating system)
- (ADM) Administrative commands
- (HW) Hardware information

These books may be ordered separately:

Using the AOM Menu System
Part Numbers: 690–23814–*nnn*
- Easy–to–use menus to use programs
- Menu manager to add, update, remove menus

Tutorial
Part Number: 690–23407–*nnn*
- Basic concepts and tasks
  Files and directories
- Utilities

International Operating System Guide
Part Number: 690–23810–*nnn*
- Character sets
- 7–bit vs. 8–bit characters

DEVELOPMENT SYSTEM

Set Part Number: 690–23417–000

Programmer's Reference (CP,S)
- (CP) Programming commands
- (S) System services, library routines

Programmer's Guide
- Lex, lint, yacc
- SCCS, make
- Extended Terminal Interface (ETI)
- Sdb, adb
- Shared libraries
- File and record locking

C Language Guide
- C User's Guide
- C Language Reference

Library Guide
- C Library Guide
- XENIX Development and Portability Guide
- International Development Guide

Developer's Guide
- DOS and OS/2 Development Guide
- STREAMS Primer
- STREAMS Programmer's Guide
- STREAMS Network Programmer's Guide

CodeView and Macro Assembler User's Guide
- The CodeView Debugger
- Macro Assembler User's Guide

Device Driver Writer's Guide
- Writing, compiling, and linking drivers
- SCSI drivers
- STREAMS and line disciplines
- (K) Kernel routines

To order any of the above manuals, call 408/434–6688, ext. 3004 and give the manual title and part number.

Operating System Documents for Different Audiences

As shown on the previous page, Altos offers many manuals with Altos UNIX System V—the manuals you receive will depend on your configuration. To help you decide which manuals are best suited to your needs, we have listed below the manuals according to three broad groups of users.

These lists are only suggested starting points in your search for information. They are not meant to imply that certain users should *not* read certain manuals. Find the user group that best applies to you, and use its list of manuals as a starting point for your reading, from which you can move on to other manuals.

Note that every Run–time System includes five manuals: the *Installation Guide*, the *User's Guide*, the *User's Reference*, the *System Administrator's Guide*, and the *System Administrator's Reference*. The Run–time System reference pages that describe the C, M, F, ADM, and HW commands ("man pages") are provided online as well. If you have the Development System, all manuals listed under "For Programmers:" come with your operating system. (All Development System reference pages are also provided online.) To order additional manuals, call (408) 434–6688, extension 3004 and give the manual title and part number.

For General Users (especially Beginners):
Tutorial
User's Guide
User's Reference (C, M, F)
Using the AOM Menu System

For System Administrators (and Advanced Users):
Installation Guide
System Administrator's Guide
System Administrator's Reference (ADM, HW)
International Operating System Guide
Programmer's Reference (CP, S)

For Programmers:
Programmer's Guide
Programmer's Reference (CP, S)
C Language Guide
Library Guide
Developer's Guide
CodeView and Macro Assembler User's Guide
Device Driver Writer's Guide

Preface

Throughout the documentation, a given command, routine, or file is referred to by its name and a section (in parentheses). For example, the programming command *cc*, is listed as *cc*(CP), which indicates that *cc* is described in the Programming Commands (CP) section.

There is a total of twelve reference sections in Altos UNIX System V, in different volumes of the Operating System and the Development System documents. (These reference sections are often called *manual pages*, or just *man pages*, in short.) For example, the *cc*(CP) command mentioned above is located in the CP section found in the *Programmer's Reference*.

This document, the *User's Reference (C, M, F)*, contains the following three reference sections:

Section	Description	Volume
C	Commands - used with the Operating System.	User's Reference
M	Miscellaneous - information used for access to devices, system maintenance, and communications.	User's Reference
F	File Formats - description of various system files not defined in section M.	User's Reference

The following table lists the remaining reference sections, the type of commands they contain, and in which document each is located.

Section	Description	Volume
ADM	Administrative Commands - used for system administration.	System Administrator's Reference
CP	Programming Commands - used with the Development System.	Programmer's Reference
DOS	DOS Cross-development subroutines and libraries	Developer's Guide
HW	Hardware device manual pages - information about hardware devices and device nodes.	System Administrator's Reference
K	Kernel routines - used for writing device drivers.	Device Driver Writer's Guide
NSL	Network Services Library - used with the STREAMS System.	Developer's Guide
S	System Calls and Library Routines - available for C and assembly language programming.	Programmer's Reference
STR	STREAMS manual pages	Developer's Guide
XNX	XENIX cross-development manual pages	Library Guide

The alphabetized table of contents following this preface lists all Altos UNIX System V commands, system calls, library routines, and file formats. In addition, in the front of each individual reference section there is an alphabetized list of all the manual pages contained in that section.

The permuted index, found at the end of the *User's Reference*, and the end of the *Programmer's Reference*, is useful in matching a desired task with the manual page that describes it. It too is an organized list of all Altos UNIX System V commands, system calls, library routines, and file formats, but organized according to function, not alphabetically.

Note that some pages in the Operating System documents refer to "include" files that are actually part of the Development System.

Alphabetized List

Commands, Systems Calls, Library Routines and File Formats

Altos UNIX® System V/386 Release 3.2

(C) Commands

Contents

Commands (C)

crypt	encode/decode
csh	invokes a shell command interpreter with C-like syntax
csplit	splits files according to context
cu	call another UNIX/XENIX system
cut	cuts out selected fields of each line of a file
date	prints and sets the date
dc	invokes an arbitrary precision calculator
dd	converts and copies a file
devnm	identifies device name
df	report number of free disk blocks
diff	compares two text files
diff3	compares three files
dircmp	compares directories
dirname	delivers directory part of pathname
disable	turns off terminals and printers
diskcp, diskcmp	copies or compares floppy disks
dos: doscat, doscp, dosdir, dosformat, dosmkdir, dosls, dosrm, dosrmdir	access to and manipulation of DOS files and DOS filesystems
dtox	change file format from MS-DOS to UNIX
dtype	determines disk type
du	summarizes disk usage
echo	echo arguments
ed, red	invokes the text editor
enable	turns on terminals and line printers
env	sets environment for command execution
ex, edit	invokes a text editor
expr	evaluates arguments as an expression
factor	factor a number
false	returns with a nonzero exit value
file	determines file type
find	finds files
finger	finds information about users
fixhdr	changes executable binary file headers
format	format floppy disks and mini-cartridge tapes
fuser	Identify processes using a file or file structure
getopt	parses command options
getopts, getoptcvt	parses command options
gets	gets a string from the standard input
greek	select terminal filter

grep, egrep, fgrep	searches a file for a pattern
grpcheck	checks group file
hd	displays files in hexadecimal format
head	prints the first few lines of a file
hello	send a message to another user
hostid	Print unique hardware ID
hp	handle special functions of Hewlett-Packard terminals
hwconfig	read the configuration information
i286emul	emulate UNIX 80286
id	prints user and group IDs and names
ismpx	return windowing terminal state
join	joins two relations
jterm	reset layer of windowing terminal
jwin	print size of layer
kill	terminates a process
ksh, rksh	KornShell, a standard/restricted command and programming language
l	lists information about contents of directory
last	indicate last logins of users and teletypes
layers	layer multiplexer for windowing terminals
lc	lists directory contents in columns
line	reads one line
ln	makes a link to a file
lock	locks a user's terminal
logname	gets login name
lp, cancel	send/cancel requests to lineprinter
lprint	print to a printer attached to the user's terminal
lpstat	print information about status of LP print service
ls	gives information about contents of directories
machid: i386	get processor type truth value
mail	interactive message processing system
man	prints reference pages in this guide
mesg	permits or denies messages sent to a terminal
mkdir	makes a directory
mknod	builds special files
mnt, umnt	mount a filesystem
more	views a file one screen full at a time
mv	moves or renames files and directories
newform	changes the format of a text file
newgrp	logs user into a new group
news	print news items
nice	runs a command at a different scheduling priority
nl	adds line numbers to a file

nohup	runs a command immune to hangups and quits
od	displays files in octal format
otar	original tape archive command
pack, pcat, unpack	compresses and expands files
passwd	change login, modem (dialup shell), filesystem, or group password
paste	merges lines of files
pax	portable archive exchange
pcpio	copy file archives in and out
pg	file perusal filter for soft-copy terminals
pr	prints files on the standard output
ps	reports process status
pscat	ASCII-to-PostScript filter
pstat	reports system information
ptar	process tape archives
purge	overwrites specified files
pwcheck	checks password file
pwd	prints working directory name
quot	summarizes file system ownership
random	generates a random number
rcp	copies files across systems
rcvtrip	notifies mail sender that recipient is away
remote	executes commands on a remote system
rm	removes files or directories
rmdir	removes directories
rsh	invokes a restricted shell (command interpreter)
sddate	prints and sets backup dates
sdiff	compares files side-by-side
sed	invokes the stream editor
setcolor, setcolour	set screen color and other screen attributes
setkey	assigns the function keys
setmode	Port modes utility
sh	invokes the shell command interpreter
shl	shell layer manager
sleep	suspends execution for an interval
sort	sorts and merges files
spell, hashmake, spellin, hashcheck	finds spelling errors
spline	interpolates smooth curve
split	splits a file into pieces
strings	find the printable strings in an object file

stty	sets the options for a terminal
su	makes the user a super-user or another user
sum	calculates checksum and counts blocks in a file
swconfig	produces a list of the software modifications to the system
tabs	set tabs on a terminal
tail	displays the last part of a file
tape, mcart	magnetic tape maintenance program
tapecntl	AT&T tape control for QIC-24/QIC-02 tape device
tapedump	dumps magnetic tape to output file
tar	archives files
tee	creates a tee in a pipe
test	tests conditions
tic	terminfo compiler
time	times a command
touch	updates access and modification times of a file
tput	queries the terminfo database
tr	translates characters
translate	translates files from one format to another
true	returns with a zero exit value
tset	provide information to set terminal modes
tty	gets the terminal's name
umask	sets file-creation mode mask
uname	prints the name of the current system
uniq	reports repeated lines in a file
units	converts units
uptime	displays information about system activity
usemouse	maps mouse input to keystrokes for use with non-mouse based programs
uucp, uulog, uuname	UNIX-to-UNIX system copy
uuencode, uudecode	encode/decode a binary file for transmission via mail
uustat	uucp status inquiry and job control
uuto, uupick	public UNIX-to-UNIX system file copy
uux	UNIX-to-UNIX system command execution
vc	version control
vi, view, vedit	invokes a screen-oriented display editor
vidi	sets the font and video mode for a video device
vmstat	report paging and system statistics
w	displays information about who is on the system and what they are doing
wait	awaits completion of background processes
wc	counts lines, words and characters

what	identifies files
who	lists who is on the system
whodo	determines who is doing what
write	writes to another user
x286emul	emulate XENIX 80286
xargs	constructs and executes commands
xprcat	use transparent printer over modem line
xtod	change file format from UNIX to MS-DOS
yes	prints string repeatedly

Intro

introduces Altos UNIX System V commands

Description

This section describes how to use many of the general-purpose commands available in the Altos UNIX System V Operating System. These command are labeled with a C, as with *date*(C). The letter "C" stands for "command."

Other commands have different labels, such as CP (for "Command Programming") or M (for "Miscellanous"). Refer to the "Preface" of this manual for a list of all the different reference sections, what commands and utilities each section describes, and in which manual each section is located.

Note that some reference sections, most notable the CP and S sections, are included only with the Development System, which is an optional supplemental package not always included with the standard Operating System.

Syntax

Unless otherwise noted, commands described in the **Syntax** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [*-option* ...] [*cmdarg* ...]

where:

[ ]	Square brackets surround an *option* or *cmdarg* that is not required.
|	A pipe (vertical bar) separates mutually exclusive options. Choose one of the items separated by this symbol.
...	Ellipses (three periods) indicate multiple occurrences of the *option* or *cmdarg*.
name	This specifies the name of an executable file.
option	(Always preceded by a "-".) *noargletter* ... or, *argletter optarg* [,...]

noargletter A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "-" (Rule 5 in the following text).

argletter A single letter representing an option requiring an option-argument.

optarg An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas or separated by white space and quoted (Rule 8 below).

cmdarg Path name (or other command argument) *not* beginning with "-", or "-" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands use them. *getopts* (C) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.

2. Command names must include only lowercase letters and digits.

3. Option names (*option* above) must be one character long.

4. All options must be preceded by "-".

5. Options with no arguments may be grouped after a single "-".

6. The first option-argument (*optarg* above) following an option must be preceded by white space.

7. Option-arguments cannot be optional.

8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., **-o xxx,z,yy** or **-o "xxx z yy"**).

9. All options must precede operands (*cmdarg* above) on the command line.

10. "--" may be used to indicate the end of the options.

11. The order of the options relative to one another should not matter.

12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.

13. "-" preceded and followed by white space should only be used to mean standard input.

See Also

getopts(C), exit(S), wait(S), getopt(S)

Diagnostics

Upon termination, each command returns 2 bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(S) and *exit*(S)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

Notes

Not all commands adhere to the syntax described here.

300, 300s

handle special functions of DASI 300 and 300s terminals

Syntax

300 [**+12**] [**-***n*] [**-dt,l,c**]

300s [**+12**] [**-***n*] [**-dt,l,c**]

Description

The *300* command supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; *300s* performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. In the following discussion of the *300* command, it should be noted that unless your system contains the text processing software, references to certain commands (e.g., *nroff*, *neqn*, *eqn*, etc.) will not work. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%. The *300* command can be used to print equations neatly, in the sequence:

neqn file ... | nroff | 300

WARNING: if your terminal has a PLOT switch, make sure it is turned *on* before *300* is used.

The behavior of *300* can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

+12 permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.

-n controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of *n* overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, *nroff* half-lines could be made to act as quarter-lines by using **-2**. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **-3** alone, having set the PITCH switch to 12-pitch.

-d*t,l,c* controls delay factors. The default setting is **-d3,90,30.** DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like **/etc/passwd.** Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output. The **-d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file **/etc/passwd** may be printed using **-d3,30,5.** The value **-d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty*(C) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

The *300* command can be used with the *nroff* **-s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

nroff -T300 files ... and nroff files ... | 300
nroff -T300-12 files ... and nroff files ... | 300 +12

The use of *300* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *300* may produce better aligned output.

See Also

450(C), mesg(C), graph(ADM), stty(C), tabs(C), tplot(ADM)

Notes

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

4014

paginator for the TEKTRONIX 4014 terminal

Syntax

4014 [**-t**] [**-n**] [**-cN**] [**-pL**] [file]

Description

The output of *4014* is intended for a TEKTRONIX 4014 terminal; *4014* arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight-space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. TELETYPE Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page, *4014* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command !*cmd* will send the *cmd* to the shell.

The command line options are:

-t Do not wait between pages (useful for directing output into a file).

-n Start printing at the current cursor position and never erase the screen.

-c*N*
Divide the screen into *N* columns and wait after the last column.

-p*L*
Set page length to *L*; *L* accepts the scale factors **i** (inches) and **l** (lines); default is lines.

See Also

pr(C)

450

handle special functions of the DASI 450 terminal

Syntax

450

Description

The *450* command supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the Diablo 1620 or Xerox 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as *300*(C). It should be noted that, unless your system contains text processing software, certain commands (e.g., *eqn*, *nroff*, *tbl*, etc.) will not work. Use *450* to print equations neatly, in the sequence:

neqn file ... | nroff | 450

WARNING: Make sure that the PLOT switch on your terminal is ON before *450* is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

Use *450* with the *nroff* **-s** flag or **.rd** requests when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the linefeed key to get any response.

In many (but not all) cases, the use of *450* can be eliminated in favor of one of the following:

nroff -T450 files ...

or

nroff -T450-12 files ...

The use of *450* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *450* may produce better aligned output.

See Also

300(C), mesg(C), stty(C), tabs(C), graph(ADM), tplot(ADM)

Notes

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

assign, deassign

assigns and deassigns devices

Syntax

assign [**-u**] [**-v**] [**-d**] [device] ...

deassign [**-u**] [**-v**] [device] ...

Description

assign attempts to assign *device* to the current user. The *device* argument must be an assignable device that is not currently assigned. An *assign* command without an argument prints a list of assignable devices along with the name of the user to whom they are assigned.

deassign is used to "deassign" devices. Without any arguments, *deassign* will deassign all devices assigned to the user. When arguments are given, an attempt is made to deassign each *device* given as an argument.

With these commands you can exclusively use a device, such as a tape drive or floppy drive. This keeps other users from using the device. They have a similar effect as *chown*(C) and *chmod*(C), although they only act on devices in **/dev**. Other aspects are discussed further on.

Available options include:

-d Performs the action of *deassign*. The **-d** option can be embedded in *device* names to assign some devices and deassign others.

-v Gives verbose output.

-u Suppresses assignment or deassignment, but performs error checking.

The *assign* command will not assign any assignable devices if it cannot assign all of them. *deassign* gives no diagnostic if the *device* cannot be deassigned. Devices can be automatically deassigned at logout, but this is not guaranteed. *Device* names can be just the beginning of the device required. For example,

```
assign fd
```

should be used to assign all floppy disk devices. Raw versions of *device* will also be assigned, e.g., the raw floppy disk devices **/dev/rfd?** would be assigned in the above example.

Note that in many installations the assignable devices such as floppy disks have general read and write access, so the *assign* command may not be necessary. This is particularly true on single-user systems. Devices supposed to be assignable with this command should be owned by the user *asg*. The directory **/dev** should be owned by **bin** and have mode 755. The *assign* command (after checking for use by someone else) will then make the device owned by whoever invokes the command, without changing the access permissions. This allows the system administrator to set up individual devices that are freely available, assignable (owned by *asg*), or nonassignable and restricted (not owned by *asg* and with some restricted mode).

Note that the first time *assign* is invoked, it builds the assignable devices table **/etc/atab**. This table is used in subsequent invocations to save repeated searches of the **/dev** directory. If one of the devices in **/dev** is changed to be assignable or unassignable (i.e., owned by *asg*), then **/etc/atab** should be removed (by the super-user) so that a correct list will be built the next time the command is invoked.

Files

/etc/atab Table of assignable devices
/dev/asglock File to prevent concurrent access

Diagnostics

Exit code 0 returned if successful, 1 if problems, 2 if *device* cannot be assigned.

at, batch

executes commands at a later time

Syntax

at time [date] [increment]

at -r job-id ...

at -l[job-id ...]

at -qletter time [date] [increment]

batch

Description

at and *batch* both accept one or more commands from the standard input to be executed at a later time. *at* and *batch* differ in the way the set of commands, or job, is scheduled: *at* allows you to specify a time when the job should be executed, while *batch* executes the job when the system load level permits. After a job is queued with either command, the program writes a job identifier (a number and a letter), along with the time the job will execute, to standard error.

at takes the following arguments:

time The *time* can be specified as 1, 2, or 4 digits. One- and two-digit numbers are taken to be hours, four digits to be hours and minutes. The time can alternately be specified as two numbers separated by a colon, meaning *hour:minute*. A suffix **am** or **pm** can be appended; otherwise a 24-hour clock time is understood. The suffix **zulu** can be used to indicate Greenwich Mean Time (GMT). The special names **noon, midnight**, and **now** are also recognized.

date An optional *date* can be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days," **today** and **tomorrow**, are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

increment

The *time* and optional *date* arguments can be modified with an increment argument of the form "+*n units*", where *n* is an integer and *units* is one of the following: **minutes, hours, days, weeks, months**, or **years**. The singular form is also accepted, and "+1 *unit*" can also be written "**next** *unit*". Thus, legitimate commands include:

at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday next week

-r *job-id ...*

Removes the specified job or jobs previously scheduled by the *at* or *batch* command. *job-id* is a job identifier returned by *at* or *batch*. Unless you are the super-user, you can only remove your own jobs.

-l [*job-id ...*]

Lists schedule times of specified jobs. If no *job-ids* are specified, lists all jobs currently scheduled for the invoking user. Unless you are the super-user, you can only list your own jobs.

-q*letter*

Places the specified job in a queue denoted by *letter*, where *letter* is any lowercase letter from "a" to "z". The queue letter is appended to the job identifier. The following letters have special significance:

a	*at* queue
b	*batch* queue
c	*cron* queue

For more information on the use of different queues, see the *queuedefs* (F) manual page.

batch takes no arguments; it submits a job for immediate execution at lower priority than an ordinary *at* job.

at and *batch* jobs are executed using *sh*(C). Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, *umask*, and *ulimit* are retained when the commands are executed. Open file descriptors, traps, and priorities are lost.

Users are permitted to use *at* and *batch* if their names appear in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** is checked to determine if a given user should be denied access to *at* and *batch*. If neither file exists, only root is allowed to submit a job. If only the **at.deny** file exists, and it is empty, global usage is permitted. The allow/deny files consist of one user name per line.

Examples

The simplest way to use *at* is to place a series of commands in a file, one per line, and execute these commands at a specified time with the following command:

at *time* < *file*

The following sequence can be used at a terminal to format the file *infile* using the text formatter *nroff*(CT), and place the output in the file *outfile*.

batch
nroff *infile* > *outfile*
⟨CTL⟩-d

The next example demonstrates redirecting standard error to a pipe (|), which is useful in a shell procedure. The file *infile* is formatted and the output placed in *outfile*, with any errors generated being mailed to *user* (output redirection is covered on the *sh*(C) manual page).

batch <<!
nroff *infile* 2>&1 >*outfile* | mail *user*
!

To have a job reschedule itself, invoke *at* from within the job. For example, if you want *shellfile* to run every Thursday, executing a series of commands and then rescheduling itself for the next Thursday, you can include code similar to the following within *shellfile*:

echo "sh *shellfile*" | at 1900 thursday next week

Files

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queuedefs	scheduling information
/usr/spool/cron/atjobs	spool area

See Also

cron(C), kill(C), mail(C), nice(C), ps(C), sh(C), queuedefs(F)

Diagnostics

Complains about syntax errors and times out of range.

Standards Conformance

at and *batch* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

auths

list and/or restrict kernel authorizations

Syntax

auths [**-v**] [**-a** authlist] [**-r** authlist] [**-c** command]

Description

auths performs actions associated with system privilege manipulation. With no arguments, *auths* returns the kernel authorizations associated with the current process. All other uses of *auths* are discussed below.

Either of the **-a** or **-r** options allow the user to alter the kernel authorizations in order to run a shell or a single command. The **-a** option requires a list of comma-separated authorizations, which become the absolute set of kernel authorizations for the new process. This new set must be a subset of the kernel authorizations of the invoking process. To start a process with a null set of kernel authorizations, use the empty string ""). The **-r** option also takes as argument a comma separated list of authorizations. These are removed from the authorization set of the invoking process when forming the kernel authorizations for the new process.

The argument to the **-c** option is passed to the user's shell as specified in the user's **/etc/passwd** entry which is run as a single command. The user's shell must support the

-c command

syntax similar to *sh*(C). When the argument is absent (and **-a** or **-r** is specified), the user's shell is invoked as a process with adjusted authorizations. Exiting that shell will resume execution in the previous shell and the original kernel authorizations will be in effect. This option may be used to run a command with restricted authorizations, i.e. fewer than those allowed the user in the protected Password database entry.

The **-v** option lists the new kernel authorizations before the new command or shell is run. It also warns with the **-a** option when more authorizations are attempted to be set than already exist or with the **-r** option when more authorizations are attempted to be removed than already exist.

The kernel authorizations are:

execsuid - allows the running of SUID programs
nopromain - does not restrict file access when running SUID programs
writeaudit - process can write directly to the audit trail
configaudit - process can change audit subsystem parameters
suspendaudit - process is not audited by the kernel
chmodsugid - process can set SUID and GID bits on files
chown - process can change file ownership

Examples

To execute a shell without the **execsuid** kernel authorization:

auths -r execsuid

To list the current kernel authorizations:

auths

To execute *yourprog* with no kernel authorizations:

auths -a "" -c yourprog

To execute *myprog* with **chmodsugid** and **execsuid**:

auths -a chmodsugid,execsuid -c myprog

See Also

sh(C), promain(M), getpriv(S), setpriv(S), getprpwent(S), "Using a Trusted System" in the *User's Guide*

awk

pattern scanning and processing language

Syntax

awk [**-F** re] [parameter...] ['prog'] [**-f** progfile] [file...]

Description

The **-F** *re* option defines the input field separator to be the regular expression *re.*

Parameters, in the form x=... y=... may be passed to *awk,* where x and y are *awk* built-in variables (see list below).

awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (') to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *file* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the **-f** *progfile* option.

Input files are read in order; if there are no files, the standard input is read. The file name - means the standard input. Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the **-F** *re* option.) The fields are denoted **$1, $2,** ...; **$0** refers to the entire line.

A pattern-action statement has the form:

pattern { action }

Either pattern or action may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations (**!**, | |, **&&**, and parentheses) of rational expressions and regular expressions. A relational expression is one of the following:

expression relop expression
expression matchop regular expression

where a relop is any of the six relational operators in C, and a matchop is either ~ (contains) or ! ~ (does not contain). A conditional is an arithmetic expression, a relational expression, the special expression

var **in** array,

or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line has been read and after the last input line has been read respectively.

Regular expressions are as in *egrep* (see *grep*(C)). In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and next occurrence of the second pattern.

A regular expression may be used to separate fields by using the **-F** *re* option or by assigning the expression to the built-in variable FS . The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

ARGC	command line argument count
ARGV	command line argument array
FILENAME	name of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression (default blank)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default **%.6g**)
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RS	input record separator (default new-line)

An action is a sequence of statements. A statement may be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
do statement while ( conditional )
for ( expression ; conditional ; expression ) statement
for ( var in array ) statement
delete array[subscript]
break
continue
{ [ statement ] ... }
expression        # commonly variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next              # skip remaining patterns on this input line
exit [expr]       # skip the rest of the input; exit status is expr
return [expr]
```

Statements are terminated by semicolons, new lines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators +, -, *, /, %, and concatenation (indicated by a blank). The **C** operators ++, --, +=, -=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The **print** statement prints its arguments on the standard output, or on a file if >*expression* is present, or on a pipe if | *cmd* is present. The arguments are separated by the current output field separator and terminated by the output record separator. The **printf** statement formats its expression list according to the format (see *printf*(S) in the *Programmer's Reference*).

awk has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: *atan2*, *cos*, *exp*, *int*, *log*, *rand*, *sin*, *sqrt*, and *srand*. *int* truncates its argument to an integer. *rand* returns a random number between 0 and 1. *srand* (expr) sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

gsub(for, repl, in)
: behaves like *sub* (see below), except that it replaces successive occurrences of the regular expression (like the *ed* global substitute command).

index(*s*, *t*) returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.

length(*s*) returns the length of its argument taken as a string, or of the whole line if there is no argument.

match(*s*, *re*) returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.

split(*s*, *a*, *fs*) splits the string s into array elements *a*[*1*], *a*[2], *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator FS if *fs* is not given.

sprintf(*fmt*, *expr*, *expr*,...)
formats the expressions according to the *printf*(S) format given by *fmt* and returns the resulting string.

sub(*for*, *repl*, *in*) substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, *awk* substitutes in the current record (**$0**).

substr(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*.

The input/output and general functions are:

close(*filename*) closes the file or pipe named *filename*.

cmd|*getline* pipes the output of *cmd* into *getline*; each successive call to *getline* returns the next line of output from *cmd*.

getline sets $0 to the next input record from the current input file.

getline <*file* sets **$0** to the next record from *file*.

getline var sets variable *var* instead.

getline var <*file* sets *var* from the next record of *file*.

system(cmd) executes *cmd* and returns to its exit status.

All forms of *getline* return 1 for successful input, 0 for end of file, and -1 for an error.

awk also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

function name(args,...) { stmts }
func name(args,...) { stmts }

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

Examples

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN       { FS = ",[ \t]*[ \t]+" }
            { print $2, $1 }
```

Add up the first column, print sum and average:

```
            { s += $1 }
END         { print "sum is",  s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Simulate *echo*(C):

```
BEGIN     {
          for (i = 1; i < ARGC; i++)
                    printf "%s", ARGV[i]
          printf "\n"
          exit
          }
```

Print file, filling in page numbers starting at 5:

```
/Page/    { $2 = n++; }
          { print }
```

command line: **awk -f program n=5 input**

See Also

grep(C), sed(C), lex(CP), printf(S)

Notes

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string (" ") to it.

Standards Conformance

awk is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

banner

prints large letters

Syntax

banner strings

Description

banner prints its arguments (each up to 10 characters long) in large letters on the standard output. This is useful for printing names at the front of printouts.

See Also

echo(C)

Standards Conformance

banner is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

basename

removes directory names from pathnames

Syntax

basename string [suffix]

Description

basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. The result is the "base" name of the file, i.e., the filename without any preceding directory path and without an extension. It is used inside substitution marks (` `) in shell procedures to construct new filenames.

The related command *dirname* deletes the last level from *string* and prints the resulting path on the standard output.

Examples

The following command displays the filename **memos** on the standard output:

```
basename /usr/johnh/memos.old .old
```

The following shell procedure, when invoked with the argument **/usr/src/cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

See Also

dirname(C), sh(C)

Standards Conformance

basename is conformant with:

The X/Open Portability Guide II of January 1987.

bc

invokes a calculator

Syntax

bc [**-c**] [**-l**] [file ...]

Description

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The **-l** argument stands for the name of an arbitrary precision math library. The syntax for *bc* programs is as follows: L means the letters a-z, E means expression, S means statement.

Comments:

Enclosed in /* and */

Names:

Simple variables: L
Array elements: L [E]
The words "base", "ibase", "obase", and "scale"; "base" and "ibase" are interchangeable.

Other operands:

Arbitrarily long numbers with optional sign and decimal point
(E)
sqrt (E)

length (E)	Number of significant decimal digits
scale (E)	Number of digits right of decimal point

L (E , ... , E)

Additive operators:

+
-

Multiplicative operators:

*
/
% (remainder)
^ (exponentiation)

Unary operators:

```
++
--          (prefix and postfix; apply to names)
```

Relational operators:

```
==
<=
>=
!=
<
>
```

Assignment operators:

```
=
=+
=-
=*
=/
=%
=^
```

Statements:

```
E
{ S ; ... ; S }
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
```

Function definitions:

```
define L ( L ,..., L ) {
        auto L, ... , L
        S; ... S
        return ( E )
}
```

Functions in -l math library:

s(x)	Sine
c(x)	Cosine
e(x)	Exponential
l(x)	Log
a(x)	Arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(C). Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

bc is actually a preprocessor for *dc*(C), which it invokes automatically, unless the **-c** (compile only) option is present. If the **-c** option is present, the *dc* input is sent to the standard output instead.

Example

The following defines a function to compute an approximate value of the exponential function:

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

The following prints the approximate values of the exponential function of the first ten integers:

```
for(i=1; i<=10; i++) e(i)
```

Files

/usr/lib/lib.bc	Mathematical library
/usr/bin/dc	Desk calculator proper

See Also

dc(C)
User's Guide

Notes

A *For* statement must have all three E's.

Quit is interpreted when read, not when executed.

Trigonometric values should be given in radians.

bdiff

compares files too large for diff(C)

Syntax

bdiff file1 file2 [n] [-s]

Description

bdiff compares two files, finds lines that are different, and prints them on the standard output. It allows processing of files that are too large for *diff*. *bdiff* splits each file into *n*-line segments, beginning with the first nonmatching lines, and invokes *diff* upon the corresponding segments. The arguments are:

n The number of lines *bdiff* splits each file into for processing. The default value is 3500. This is useful when 3500-line segments are too large for *diff*.

-s Suppresses printing of *bdiff* diagnostics. Note that this does not suppress printing of diagnostics from *diff*.

If *file1* (or *file2*) is a dash (-), the standard input is read.

The output of *bdiff* is exactly that of *diff*. Line numbers are adjusted to account for the segmenting of the files, and the output looks as if the files had been processed whole.

Files

/tmp/bd?????

See Also

diff(C)

Notes

Because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

Specify the maximum number of lines if the first difference is too far down in the file for *diff* and an error is received.

bfs

scans big files

Syntax

bfs [**-**] name

Description

bfs is like *ed* (C) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 255 characters per line. *bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(C) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional dash (-) suppresses printing of sizes. Input is prompted for with an asterisk (*) when "P" and RETURN are typed. The "P" acts as a toggle, so prompting can be turned off again by entering another "P" and a RETURN. Note that messages are given in response to errors only if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols other than the standard slash (/) and (?): A greater-than sign (>) indicates downward search without wraparound, and a less-than sign (<) indicates upward search without wraparound. Note that parentheses and curly braces are special and need to be escaped with a backslash (\). Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regex* (S)). Differences between *ed* and *bfs* are listed below:

+ A regular expression followed by + means *one or more times*. For example, **[0-9]+** is equivalent to **[0-9][0-9]***.

\{m\} \{m,\} \{m,u\}
Integer values enclosed in **\{ \}** indicate the number of times the preceding regular expression is to be applied. *m* is the minimum number and *u* is a number, less than 256, which is the maximum. If only *m* is present (e.g., \{m\}), it indicates the exact number of times the regular expression is to be applied. \{m,\} is analogous to \{m,infinity\}. The plus (+) and star (*) operations are equivalent to \{1,\} and \{0,\} respectively.

(...)$*n* The value of the enclosed regular expression is to be returned. The value will be stored in the *(n+1)*th argument following the subject argument. At most ten enclosed regular expressions are allowed. *regex* makes its assignments unconditionally.

(...) Parentheses are used for grouping. An operator, e.g. *, +, \{ \}, can work on a single character or a regular expression enclosed in parenthesis. For example, \(a*\(cb+\)*\)$0.

There is also a slight difference in mark names: only the letters ''a'' through ''z'' may be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **p**, **q**, **w**, **=**, **!** and null commands operate as described under *ed* except that **e** doesn't remember filenames and **g** and **v** when given no arguments return the line after the line you were on. Commands such as **---**, **+++-**, **+++=**, **-12**, and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no remembered filename. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt** and **xc** commands, below). The following additional commands are available:

xf *file*

Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received, or an error occurs, reading resumes with the file containing the **xf**. **xf** commands may be nested to a depth of 10.

xo [*file*]

Further output from the **p** and null commands is diverted to the named *file*. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label*

This positions a *label* in a command file. The *label* is terminated by a newline, and blanks between the **:** and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(. , .)**xb**/*regular expression*/*label*

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.

2. The second address is less than the first.

3. The regular expression doesn't match at least one line in the specified range, including the first and last lines.

On success, dot (.) is set to the line matched and a jump is made to *label*. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.

The **xb** command is allowed only if it is read from somewhere other than a terminal. If it is read from a pipe only a downward jump is possible.

xt *number*

Output from the **p** and null commands is truncated to a maximum of *number* characters. The initial number is 255.

xv[*digit*] [*spaces*] [*value*]

The variable name is the specified *digit* following the **xv**. **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. **xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6**:

```
1,%5p
1,%5
%6
```

prints the first 100 lines.

```
g/%5/p
```

globally searches for the characters **100** and prints each line containing a match. To escape the special meaning of **%**, a \ must precede it. For example,

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* characters, decimal integers, or strings.

Another feature of the **xv** command is that the first line of output from a Altos UNIX System V command can be stored into a variable.

The only requirement is that the first character of *value* be a !. For example,

```
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

puts the current line in variable **5**, prints it, and increments the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a \. For example,

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*

xbn *label*
These two commands test the last saved *return code* from the execution of an Altos UNIX System V command (**!***command*) or nonzero value, respectively, and jump to the specified label. The two examples below search for the next five lines containing the string **size**:

```
xv55
:1
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn 1
xv45
:1
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz 1
```

xc [*switch*]
If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0**, it is not. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

See Also

csplit(C), ed(C), umask(C)

Diagnostics

? for errors in commands if prompting is turned off. Self-explanatory error messages when prompting is on.

cal

prints a calendar

Syntax

cal [[month] year]

Description

cal prints a calendar for the specified year. If a month is also specified, a calendar for that month only is printed. If no arguments are specified, the current, previous, and following months are printed, along with the current date and time. The *year* must be a number between 1 and 9999; *month* must be a number between 1 and 12 or enough characters to specify a particular month. For example, **May** must be given to distinguish it from March, but **S** is sufficient to specify September. If only a month string is given, only that month of the current year is printed.

Notes

Beware that "cal 84" refers to the year 84, not 1984.

The calendar produced is that for England and her colonies. Note that England switched from the Julian to the Gregorian calendar in September of 1752, at which time eleven days were excised from the year. To see the result of this switch, try "cal 9 1752".

Standards Conformance

cal is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

calendar

invokes a reminder service

Syntax

calendar [-]

Description

calendar consults the file **calendar** in the user's current directory and mails him lines that contain today's or tomorrow's date. Most reasonable month-day dates, such as "Sep. 14," "september 14", and "9/14", are recognized, but not "14 September", or "14/9".

On weekends "tomorrow" extends through Monday. Lines that contain the date of a Monday will be sent to the user on the previous Friday. This is not true for holidays.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in his login directory and sends the result to the standard output. Normally this is done daily, in the early morning, under the control of *cron* (C).

Files

calendar

/usr/lib/calprog To figure out today's and tomorrow's dates

/etc/passwd

/tmp/cal*

See Also

cron(C), mail(C)

Notes

To get reminder service, a user's **calendar** file must have read permission for all.

Standards Conformance

calendar is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

cat

concatenates and displays files

Syntax

cat [**-u**] [**-s**] [**-v**] [**-t**] [**-e**] file ...

Description

cat reads each *file* in sequence and writes it on the standard output. If no input file is given, or if a single dash (-) is given, *cat* reads from the standard input. The options are:

-s Suppresses warnings about nonexistent files.

-u Causes the output to be unbuffered.

-v Causes non-printing characters (with the exception of tabs, new-lines, and form feeds) to be displayed. Control characters are displayed as "^X" (Ctrl-*X*), where *X* is the key pressed with the Ctrl key (for example, Ctrl-M is displayed as ^M). The DEL character (octal 0177) is printed as "^?." Non-ASCII characters (with the high bit set) are printed as "M -x," where *x* is the character specified by the seven low order bits.

-t Causes tabs to be printed as "^I" and form feeds as "^L". This option is ignored if the **-v** option is not specified.

-e Causes a "$" character to be printed at the end of each line (prior to the new-line). This option is ignored if the **-v** option is not set.

No input file may have the same name as the output file unless it is a special file.

Examples

The following example displays *file* on the standard output:

```
cat file
```

The following example concatenates *file1* and *file2* and places the result in *file3*:

```
cat file1 file2 >file3
```

The following example concatenates *file1* and appends it to *file2*:

```
cat file1 >> file2
```

See Also

cp(C), pr(C)

Warning

Command lines such as:

```
cat file1 file2 > file1
```

will cause the original data in *file1* to be lost; therefore, you must be careful when using special shell characters.

Standards Conformance

cat is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

cd

changes working directory

Syntax

cd [directory]

Description

If specified, *directory* becomes the new working directory; otherwise the value of the shell parameter $HOME is used. The process must have search (execute) permission in all directories (components) specified in the full pathname of *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and executed by the shell.

If the shell is reading its commands from a terminal, and the specified directory does not exist (or some component cannot be searched), spelling correction is applied to each component of *directory*, in a search for the ''correct'' name. The shell then asks whether or not to try and change directory to the corrected directory name; an answer of *n* means ''no'', and anything else is taken as ''yes''.

Notes

Wildcard designators will work with the **cd** command.

See Also

pwd(C), sh(C), chdir(S)

checkmail

checks for mail which has been submitted but not delivered

Syntax

checkmail [-a] [-f] [-m]

Description

checkmail checks the mail queue on the local machine for messages which have been sent by the invoker. If invoked without any arguments, the "Subject:" of each message found is given along with a list of addressees that have not yet received the message. Usually, messages are still in the queue because the addressee's host is down.

The **-a** (all addresses) option causes all addresses to be shown (both delivered and undelivered). Some delivered addresses may not appear since some sites prune already delivered addresses from the address list files for efficiency. The **-f** (fast) option suppresses the printing of the "Subject" line. The **-m** (all messages) option causes checkmail to check all messages in the mail queue, not just those of the invoker. This is only useful for mail system maintainers who wish to find obstinate hosts.

See Also

send(ADM), deliver(ADM), mmdf(ADM)

chgrp

changes group ID

Syntax

chgrp group file ...

Description

chgrp changes the group ID of each *file* to *group*. The group may be either a decimal group ID or a group name found in the file **/etc/group.**

Files

/etc/passwd

/etc/group

See Also

chown(C), chown(S), passwd(F), group(F)

Notes

Only the owner or the super-user can change the group ID of a file.

Standards Conformance

chgrp is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

chmod

changes the access permissions of a file or directory

Syntax

chmod mode file
chmod [who] [+|-|=] [permission ...] file ...

Description

The *chmod* command changes the access permissions (or *mode*) of a specified file or directory. It is used to control file and directory access by users other than the owner and super-user. The *mode* may be an expression composed of letters and operators (called *symbolic mode*), or a number (called *absolute mode*).

A *chmod* command using *symbolic mode* has the form:

chmod [who] [+|-|=] [permission ...] filename

In place of *who* you can use one or any combination of the following letters:

a Stands for "all users". If *who* is not indicated on the command line, **a** is the default. The definition of "all users" depends on the user's *umask*. See *umask*(C).

g Stands for "group", all users who have the same group ID as the owner of the file or directory.

o Stands for "others", all users on the system.

u Stands for "user", the owner of the file or directory.

The operators are:

+ Adds permission

- Removes permission

= Assigns the indicated permission and removes all other permissions (if any) for that *who*. If no permission is assigned, existing permissions are removed.

Permissions can be any combination of the following letters:

x Execute (search permission for directories)

r Read

w Write

s Sets owner or group ID on execution of the file to that of the owner of the file. The mode "u+s" sets the user ID bit for the file. The mode "g+s" sets the group ID bit. Other combinations have no effect. When the group ID bit is set on a directory, all files created under it thereafter receive the group ID of that directory. When the group ID bit is not set, files are created with the group ID of the creating process/user.

t This is known as the "sticky bit." (see *chmod*(S)). Only the mode "u+t" sets the sticky bit. All other combinations have no effect. When this bit is set on a directory, files within the directory cannot be removed by anyone but the owner or the super-user. The owner can set or remove the sticky bit.

l Mandatory locking will occur during access

Multiple symbolic modes may be given, separated by commas, on a single command line. See the following Examples section for sample permission settings.

Mandatory file and record locking refers to a file having locked reading or writing permissions while a program is accessing that file. A file cannot have group execution permission and be able to be locked on execution. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. The following examples show illegal uses of *chmod* and will generate error messages:

chmod g+x,+l *filename*

chmod g+s,+l *filename*

A *chmod* command using *absolute mode* has the form:

chmod *mode filename*

where *mode* is an octal number constructed by performing logical OR on the following:

4000 Set user ID on execution

20#0 Set group ID on execution if "#" is 7, 5, 3, or 1 and enable mandatory locking if "#" is 6, 4, 2, or 0.

1000	Sets the sticky bit (see *chmod*(S))
0400	Read by owner
0200	Write by owner
0100	Execute (search in directory) by owner
0040	Read by group
0020	Write by group
0010	Execute (search in directory) by group
0004	Read by others
0002	Write by others
0001	Execute (search in directory) by others
0000	No permissions

Examples

Symbolic Mode

The following command gives all users execute permission for *file*:

chmod +x *file*

The following command removes read and write permission for group and others from *file*:

chmod go-rw *file*

The following command gives other users read and write permission for *file*:

chmod o+rw *file*

The following command gives read permission to group and other:

chmod g+r,o+r *file*

Absolute Mode

The following command gives all users read, write and execute permission for *file*:

chmod 0777 *file*

The following command gives read and write permission to all users for *file*:

chmod 0666 *file*

The following command gives read and write permission to the owner of *file* only:

chmod 0600 *file*

The following example causes the file to be locked on access:

chmod +l *file*

See Also

ls(C), chmod(S)

Notes

The setuid, setgid and sticky bit settings have no effect on shell scripts.

Standards Conformance

chmod is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

chown

changes owner ID

Syntax

chown owner file ...

Description

chown changes the owner ID of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the file **/etc/passwd.**

Files

/etc/passwd

/etc/group

See Also

chgrp(C), chown(S), group(F), passwd(F)

Notes

Use of this utility is governed by the **chown** kernel authorization. If this authorization is not granted, ownership of files can only be changed by **root**. Restricted *chown* is required for NIST FIPS 151-1 conformance. The **chown** authorization should not be assigned to users if you wish to conform to these requirements.

Standards Conformance

chown is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
NIST FIPS 151-1;
and The X/Open Portability Guide II of January 1987.

clear

clears a terminal screen

Syntax

clear [term]

Description

The *clear* command clears the screen. If *term* is not specified, the terminal type is obtained from the **TERM** environment variable.

If a video terminal does not have a clear screen capability, newlines are output to scroll the screen clear. If the terminal is a hardcopy, the paper is advanced to the top of the next page.

Files

/etc/termcap

See Also

environ(M), termcap(F), tput(C)

Notes

If the standard output is not a terminal, *clear* issues an error message.

cmchk

reports hard disk block size

Syntax

cmchk

Description

Reports the hard disk block size in 512-byte blocks.

Value Added

cmchk is an extension of AT&T System V provided by Altos UNIX System V.

cmp

compares two files

Syntax

cmp [**-l**] [**-s**] file1 file2

Description

cmp compares two files and, if they are different, displays the byte and line number of the differences. If *file1* is -, the standard input is used.

The options are:

-l Prints the byte number (decimal) and the differing bytes (octal) for each difference.

-s Returns an exit code only, 0 for identical files, 1 for different files and 2 for an inaccessible or missing file.

This command should be used to compare binary files; use *diff*(C) or *diff3*(C) to compare text files.

See Also

comm(C), diff(C), diff3(C)

Standards Conformance

cmp is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

col

filters reverse linefeeds

Syntax

col [**-bfxp**]

Description

col prepares output from processes, such as the text formatter *nroff*(CT), for output on devices that limit or do not allow reverse or half-line motions. *col* is typically used to process nroff output text that contains tables generated by the **tbl** program. A typical command line might be:

tbl *file* | nroff | col | lpr

col takes the following options:

-b Assumes the output device in use is not capable of backspacing. If two or more characters appear in the same place, *col* outputs the last character read.

-f Allows forward half linefeeds. If not given, *col* accepts half line motions in its input, but text that would appear between lines is moved down to the next full line. Reverse full and half linefeeds are never allowed with this option.

-x Prevents conversion of whitespace to tabs on output. *col* normally converts whitespace to tabs wherever possible to shorten printing time.

-p Causes *col* to ignore unknown escape sequences found in its input and pass them to the output as regular characters. Because these characters are subject to overprinting from reverse line motions, the use of this option is discouraged unless the user is fully aware of the position of the escape sequences.

col assumes that the ASCII control characters **SO** (octal 016) and **SI** (octal 017) start and end text in an alternate character set. If you have a reverse linefeed (**ESC 7**), reverse half linefeed (**ESC 8**), or forward half linefeed (**ESC 9**), within an **SI-SO** sequence, the **ESC 7, 8** and **9** are still recognized as line motions.

On input, the only control characters *col* accepts are space, backspace, tab, return, newline, reverse linefeed (**ESC 7**), reverse half linefeed (**ESC 8**), forward half linefeed (**ESC 9**), alternate character start(**SI**), alternate character end (SO), and vertical tag (VT). (The **VT** character is an alternate form of full reverse linefeed, included for compatibility with some earlier programs of this type.) All other nonprinting characters are ignored.

See Also

nroff(CT), tbl(CT)

Notes

col cannot back up more than 128 lines.

col allows at most 800 characters, including backspaces, on a line.

Vertical motions that would back up over the first line of the document are ignored. Therefore, the first line must not contain any superscripts.

Standards Conformance

col is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

comm

selects or rejects lines common to two sorted files

Syntax

comm [**-123**] file1 file2

Description

comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort* (C)), and produces a three-column output: lines only in *file1* ; lines only in *file2* ; and lines in both files. The filename - means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm -12** prints only the lines common to the two files; **comm -23** prints only lines in the first file but not in the second; **comm -123** is a no-op.

See Also

cmp(C), diff(C), sort(C), uniq(C)

Standards Conformance

comm is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

compress, uncompress, zcat

compress data for storage, uncompress and display compressed files

Syntax

compress [**-dfFqc**] [**-b** *bits*] *file*
uncompress [**-fqc**] *file*
zcat *file*

Description

compress takes a file and compresses it to the smallest possible size, creates a compressed output file, and removes the original file unless the **-c** option is present. Compression is achieved by encoding common strings within the file. *uncompress* restores a previously compressed file to its uncompressed state and removes the compressed version. *zcat* uncompresses and displays a file on the standard output.

If no file is specified on the command line, input is taken from the standard input and the output is directed to the standard output. Output defaults to a file with the same filename as the input file with the suffix ".Z" or it can be directed through the standard output. The output files have the same permissions and ownership as the corresponding input files or the user's standard permissions if output is directed through the standard output.

If no space is saved by compression, the output file is not written unless the **-F** flag is present on the command line.

Options

The following options are available from the command line:

-d Decompresses a compressed file.

-c Writes output on the standard output and does not remove original file.

-b*bits* Specifies the maximum number of bits to use in encoding.

-f Overwrites previous output file.

-F Writes output file even if compression saves no space.

-q Generates no output except error messages, if any.

See Also

pack(C), ar(C), tar(C), cat(C)

copy

copies groups of files

Syntax

copy [option] ... source ... dest

Description

The *copy* command copies the contents of directories to another directory. It is possible to copy whole file systems since directories are made when needed.

If files, directories, or special files do not exist at the destination, then they are created with the same modes and flags as the source. In addition, the super-user may set the user and group ID. The owner and mode are not changed if the destination file exists.

Note that there may be more than one source directory. If so, the effect is the same as if the *copy* command had been issued for each source directory with the same destination directory for each copy.

Options do not have to be given as separate arguments, and may appear in any order, even after the other arguments. The options are:

-a Asks the user before attempting a copy. If the response does not begin with a "y", then a copy is not done. When used together with the **-v** option, it overrides the verbose option so that messages regarding the copy action are not displayed.

-l Uses links instead whenever they can be used. Otherwise a copy is done. Note that links are never done for special files or directories.

-n Requires the destination file to be new. If not, then the *copy* command does not change the destination file. The **-n** flag is meaningless for directories. For special files a **-n** flag is assumed (i.e., the destination of a special file must not exist).

-o If set then every file copied has its owner and group set to those of the source. If not set, then the file's owner is the user who invoked the program.

-m If set, then every file copied has its modification time and access time set to that of the source. If not set, then the modification time is set to the time of the copy.

-r If set, then every directory is recursively examined as it is encountered. If not set then any directories that are found are ignored.

-ad Asks the user whether a **-r** flag applies when a directory is discovered. If the answer does not begin with a "y", then the directory is ignored.

-v Messages are printed that reveal what the program is doing. If used with the **-a** option, the **-a** option is given priority so that it overrides the verbose option, and the copy action message is not displayed.

Arguments to *copy* are:

source This may be a file, directory or special file. It must exist. If it is not a directory, then the results of the command are the same as for the *cp* command.

dest The destination must be either a file or directory name that is different from the source.

If the source and destination are anything but directories, then *copy* acts just like a *cp* command. If both are directories, then *copy* copies each file into the destination directory according to the flags that have been set.

Examples

This command line verbosely copies all files in the current directory to **/tmp/food**:

```
copy -v . /tmp/food
```

The next command line copies all files, except for those that begin with a period (.), and copies the immediate contents of any child directories:

```
copy * /tmp/logic
```

This command is the same as the previous one, except that it recursively examines all subdirectories, and it sets group and ownership permissions on the destination files to be the same as the source files:

```
copy -ro * /tmp/logic
```

Notes

Special device files can be copied. When they are copied, any data associated with the specified device is *not* copied.

cp

copies files

Syntax

cp file1 file2

cp files directory

Description

There are two ways to use the *cp* command. With the first way, *file1* is copied to *file2*. Under no circumstance can *file1* and *file2* be identical. With the second way, *directory* is the location of a directory into which one or more *files* are copied. This directory must exist prior to the execution of the *cp* command.

See Also

copy(C), cpio(C), ln(C), mv(C), rm(C), chmod(S)

Notes

Special device files can be copied. If the file is a named pipe, then the data in the pipe is copied to a regular file. Similarly, if the file is a device, then the file is read until the end-of-file is reached, and that data is copied to a regular file. It is not possible to copy a directory to a file.

Standards Conformance

cp is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

cpio

copy file archives in and out

Syntax

cpio -o [**acBvV**] [**-C** bufsize] [[**-O** file] [**-K** volumesize] [**-M** message]]

cpio -i [**BcdmrtTuvVfsSb6k**] [**-C** bufsize] [[**-I** file] [**-K** volumesize] [**-M** message]] [pattern ...]

cpio -p [**adlmuvV**] directory

Description

cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary by default.

cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *patterns* are regular expressions given in the filename-generating notation of *sh*(C). In *patterns*, metacharacters ?, *, and [...] match the slash (/) character, and backslash (\) is an escape character. A ! metacharacter means *not*. (For example, the **!abc*** pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). Each *pattern* must be enclosed in double quotes otherwise the name of a file in the current directory is used. Extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous *cpio -o* . The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous *cpio -o* . NOTE: If *cpio -i* tries to create a file that already exists and the existing file is the same age or newer, *cpio* will output a warning message and not replace the file. (The **-u** option can be used to unconditionally overwrite the existing file.)

cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below. Archives of text files created by *cpio* are portable between implementations of UNIX System V.

The meanings of the available options are:

-a Reset *access* times of input files after they have been copied. Access times are not reset for linked files when *cpio -pla* is specified.

-b Reverse the order of the *bytes* within each word. Use only with the **-i** option.

-B Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this and the **-C** options are not used. (**-B** does not apply to the *pass* option; **-B** is meaningful only with data directed to or from a character-special device, e.g., **/dev/rdsk/f0q15dt**.)

-c Write header information in ASCII *character* form for portability. Always use this option when origin and destination machines are different types.

-C *bufsize*
Input/output is to be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this and **-B** options are not used. (**-C** does not apply to the *pass* option; **-C** is meaningful only with data directed to or from a character-special device, e.g., **/dev/rmt/c0s0**.) When used with the **-K** option, *bufsize* is forced to be a 1K multiple.

-d *directories* are to be created as needed.

-f Copy in all *files* except those in *patterns*. (See the paragraph on *cpio -i* for a description of *patterns*.)

-I *file*
Read the contents of *file* as input. If *file* is a character-special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **-i** option.

-k Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For *cpio* archives that contain other *cpio* archives, if an error is encountered, *cpio* may terminate prematurely. *cpio* will find the next good header, which may be one for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the **-i** option.

-l Whenever possible, *link* files rather than copying them. Usable only with the **-p** option.

-m
Retain previous file *modification* time. This option is ineffective on directories that are being copied.

-K *volumesize*
Specifies the size of the media volume. Must be in 1K blocks. For example, a 1.2 MB floppy disk has a *volumesize* of 1200. Must include the **-C** option with a *bufsize* multiple of 1K.

-M *message*
Define a message to use when switching media. When you use the **-O** or **-I** options and specify a character-special device, you can use this option to define the message that is printed when you reach the end of the medium. One **%d** can be placed in the message to print the sequence number of the next medium needed to continue.

-O *file*
Direct the output of cpio to *file*. If *file* is a character-special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **-o** option.

-r Interactively *rename* files. If the user types a null line, the file is skipped. If the user types a ".", the original pathname will be copied. (Not available with *cpio -p*.)

-s *swap* bytes within each half word. Use only with the **-i** option.

-S *Swap* halfwords within each word. Use only with the **-i** option.

-T Truncate long filenames to 14 characters. Use only with the **-i** option.

-t Print a *table of contents* of the input. No files are created.

-u Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).

-v *verbose*: causes a list of file names to be printed. When used with the **-t** option, the table of contents looks like the output of an **ls -l** command [see *ls*(C)].

-V *Special* Verbose: print a dot for each file seen. Useful to assure the user that *cpio* is working without printing out all file names.

-6 Process an old (i.e., UNIX System *Sixth* Edition format) file. Use only with the **-i** option.

NOTE: *cpio* assumes 4-byte words.

If *cpio* reaches end of medium (end of a diskette for example) when writing to (**-o**) or reading from (**-i**) a character-special device, and **-O** and **-I** are not used, *cpio* will print the message:

If you want to go on, type device/file name when ready.

To continue, you must replace the medium and type the character-special device name (**/dev/rdsk/f0q15dt** for example) and a carriage return. You may want to continue by directing *cpio* to use a different device. For example, if you have two floppy drives, you may want to switch between them so *cpio* can proceed while you are changing the floppies. (A carriage return alone causes the *cpio* process to exit.)

Examples

The following examples show three uses of *cpio.*

When standard input is directed through a pipe to **cpio -o**, it groups the files so they can be directed (>) to a single file (*../newfile*). The **-c** option insures that the file will be portable to other machines. Instead of *ls*(C), you could use *find*(C), *echo*(C), *cat*(C), etc., to pipe a list of names to *cpio*. You could direct the output to a device instead of a file.

ls | cpio -oc >*../newfile*

cpio -i uses the output file of **cpio -o** (directed through a pipe with **cat** in the example), extracts those files that match the patterns (**memo/a1**, **memo/b***), creates directories below the current directory as needed (**-d** option), and places the files in the appropriate directories. The **-c** option is used when the file is created with a portable header. If no patterns were given, all files from *newfile* would be placed in the directory.

cat newfile | cpio -icd *"memo/a1" "memo/b*"*

cpio -p takes the file names piped to it and copies or links (**-l** option) those files to another directory on your machine (*newdir* in the example). The **-d** options says to create directories as needed. The **-m** option says retain the modification time. [It is important to use the **-depth** option of *find*(C) to generate path names for *cpio*. This eliminates problems *cpio* could have trying to create files under read-only directories.]

find . -depth -print | cpio -pdlmv *newdir*

See Also

cat(C), echo(C), find(C), ls(C), tar(C), cpio(F)

Notes

1) Path names are restricted to 256 characters.
2) Only the super-user can copy special files.
3) Blocks are reported in 512-byte quantities.
4) If a file has 000 permissions, contains more than 0 characters of data, and the user is not root, the file will not be saved or restored.

Standards Conformance

cpio is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

cron

executes commands scheduled by at, batch, and crontab

Syntax

/etc/cron

Description

cron is the clock daemon that executes commands at specified dates and times. *cron* processes jobs submitted with *at*(C), *batch*(C), and - *crontab*(C). *cron* never exits; the *cron* command usually appears in the **/etc/rc2** scripts to be invoked by *init*(M) when the system is brought up in multi-user mode.

Files

/etc/default/cron	cron logging default information
/usr/lib/cron	main cron directory
/usr/lib/cron/atjobs	at directory
/usr/spool/cron/crontabs	crontab directory
/usr/lib/cron/log	accounting information
/usr/lib/cron/queuedefs	cron data file
/usr/lib/cron/.proto	cron environment information

See Also

at(C), crontab(C), queuedefs(F), sh(C)

Diagnostics

A history of all actions by *cron* can be recorded in **/usr/lib/cron/log**. This logging occurs only if the variable CRONLOG is set to YES in **/etc/default/cron**. By default this value is set to NO and no logging occurs. If logging should be turned on, be sure to check the size of the log file regularly.

Standards Conformance

cron is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

crontab

schedule commands to be executed at regular intervals

Syntax

crontab [file]
crontab -r
crontab -l

Description

The *crontab* command can be used to schedule commands to be executed at regular intervals. These commands are stored in the user's crontab file, **/usr/spool/cron/crontabs/***username*. Any output or errors generated by the commands are mailed to the user.

If called with no options, *crontab* copies the specified file, or standard input if no file is specified, into the **crontabs** directory (if the user has a previous crontab file, it is replaced).

The **-r** option removes the user's crontab file from the crontab directory.

The **-l** option lists the contents of the user's crontab file.

If the file **/usr/lib/cron/cron.allow** exists, only the users listed in that file are allowed to use *crontab*. If **cron.allow** does not exist, and the file **/usr/lib/cron/cron.deny** does, then all users not listed in **cron.deny** are allowed access to *crontab*, with an empty **cron.deny** allowing global usage. If neither file exists, only the super user is allowed to submit a job. The allow/deny files consist of one user name per line.

The **crontabs** files consist of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (0-6, with 0=Sunday). Each of these patterns may contain:

- A number in the (respective) range indicated above

- Two numbers separated by a minus (indicating an inclusive range)

- A list of numbers separated by commas (meaning all of these numbers)

- An asterisk (meaning all legal values)

Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field is a string that is executed by the shell at the specified time(s). A % in this field is translated into a newline character. Only the first line (up to a % or end-of-line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh.** Users who desire to have their *.profile* executed must explicitly do so in the crontab file. *cron* supplies a default environment for every shell, defining **HOME, LOGNAME, SHELL (=/bin/sh),** and **PATH (=/bin:/usr/bin:).**

Examples

An example **crontabs** file follows:

```
0        4  *  *  *  calendar -
15       4  *  *  *  find /usr/preserve -mtime +7 -exec rm -f {} ;
30       4  1  *  1  /usr/lib/uucp/uuclean
40       4  *  *  *  find / -name '#*' -atime +3 -exec rm -f {} ;
1,21,41  *  *  *  *  (echo -n ' '; date; echo ) >/dev/console
```

The lines in this example do the following: run the calendar program every night at 4:00 am, clear old files from the **/etc/preserve** directory every night at 4:15 am, clean up the uucp spool directory every Monday and the first of every month at 4:30 am, find and remove any old files with names beginning with "#" every night at 4:40 am, and echo the current date and time to the console three times an hour at one minute, 21 minutes, and 41 minutes past the hour.

Files

/usr/lib/cron	main cron directory

/usr/spool/cron/crontabs	crontab directory
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users
/usr/lib/cron/.proto	cron environment information
/usr/lib/cron/queuedefs	cron data file

See Also

at(C), cron(C), sh(C)

Diagnostics

crontab exits and returns a value of 55 if it cannot allocate enough memory. If it exits for any other reason, it returns a value of 1.

Notes

crontab commands are executed by *cron*(C). *cron* reads the files in the **crontabs** directory only on startup or when a new crontab is submitted with the *crontab* command, so changes made to these files by hand will not take effect until the system is rebooted. Changes submitted with the *crontab* command will take effect as soon as *cron* is free to read them (that is, when *cron* is not in the process of running a scheduled job or reading another newly submitted *at*(C) or *crontab* job.).

Users who do not wish to have output from their commands mailed to them may want to redirect it to a file:

```
0  *  *  *  *  who >> /tmp/whofile 2> /dev/null
```

The example above would append the output of the *who*(C) command to a file, and throw away any errors generated. For more details on output redirection, see the *sh*(C) manual page.

Users should remember to redirect the standard output and standard error of their commands otherwise any generated output or errors will be mailed to the user.

crontab will overwrite any previous crontab submitted by the same user.

Standards Conformance

crontab is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

crypt

encode/decode

Syntax

crypt [password]
crypt [**-k**]

Description

The *crypt* command reads from the standard input and writes to the standard output. The *password* is a key that selects a particular transformation. If no argument is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. If the **-k** option is used, *crypt* will use the key assigned to the environment variable CRYPTKEY. The *crypt* command encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

Files encrypted by *crypt* are compatible with those treated by the editors *ed*(C), *edit*(C), *ex*(C), and *vi*(C) in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or clear text can become visible must be minimized.

The *crypt* command implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e., to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

If the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(C) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of *crypt*.

Files

/dev/tty for typed key

See Also

ed(C), edit(C), ex(C), makekey(C), ps(C), stty(C), vi(C)

Notes

If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

Distribution of the crypt libraries and utilities is regulated by the U.S. Government and are not available to sites outside of the United States and its territories. Because we cannot control the destination of the software, these utilities are not included in the standard product. If your site is within the U.S. or its territories, you can obtain the *crypt* software through your product distributor or reseller.

csh

invokes a shell command interpreter with C-like syntax

Syntax

csh [**-cefinstvVxX**] [arg ...]

Description

csh is a command language interpreter. It begins by executing commands from the file **.cshrc** in the home directory of the invoker. If this is a login shell, it also executes commands from the file **.login** there. In the normal case, the shell begins reading commands from the terminal, prompting with % . Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally, each command in the current line is executed.

When a login shell terminates, it executes commands from the file **.logout** in the user's home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters &, |, ;, <, >, (,), form separate words. If doubled in &&, | |, <<, or >>, these pairs form single words. These parser metacharacters may be made part of other words, or their special meaning prevented, by preceding them with \ A newline preceded by a \ is equivalent to a blank.

In addition, strings enclosed in matched pairs of quotations, ´, ` or ", form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of ´ or " characters, a newline preceded by a \ gives a true newline character.

When the shell's input is not a terminal, the character # introduces a comment which continues to the end of the input line. It does not have this special meaning when preceded by \ and placed inside the quotation marks `, ´, or ".

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by | characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ;, and are then executed sequentially. A sequence of pipelines may be executed without waiting for it to terminate by following it with a &. Such a sequence is automatically prevented from being terminated by a hangup signal; the *nohup* command need not be used.

Any of the above may be placed in parentheses to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with | | or && indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions.)*

Substitutions

The following sections describe the various transformations the shell performs on the input in the order in which they occur.

History Substitutions

History substitutions can be used to reintroduce sequences of words from previous commands, possibly performing modifications on these words. Thus, history substitutions provide a generalization of a *redo* function.

History substitutions begin with the character ! and may begin **anywhere** in the input stream if a history substitution is not already in progress. The ! may be preceded by a \ to prevent its special meaning; a ! is passed unchanged when it is followed by a blank, tab, newline, =, or (. History substitutions may also occur when an input line begins with ^. This special abbreviation will be described later.

Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been entered without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list, the size of which is controlled by the *history* variable. The previous command is always retained. Commands are numbered sequentially from 1.

For example, enter the command:

history

Now, consider the following output from the history command:

```
 9  write michael
10  ex write.c
11  cat oldwrite.c
12  diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the prompt by placing a ! in the prompt string.

With the current event 13 we can refer to previous events by event number !11, relatively as in !-2 (referring to the same event), by a prefix of a command word as in !d for event 12 or !w for event 9, or by a string contained in a word in the command as in !?mic? also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case !! refers to the previous command; thus !! alone is essentially a *redo*. The form !# references the current command (the one being entered). It allows a word to be selected from further left in the line, to avoid retyping a long name, as in !#:1.

To select words from an event, we can follow the event specification by a : and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, and so on. The basic word designators are:

0 First (command) word

n *n*th argument

^ First argument, i.e. 1

$ Last argument

% Word matched by (immediately preceding) ?*s*? search

x-*y*
Range of words

-*y* Abbreviates 0-*y*

* Abbreviates ^-$, or nothing if only 1 word in event

x * Abbreviates *x* -$

x - Like *x* * but omitting word $

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ^, $, *, - or %. After the optional word designator, a sequence of modifiers can be placed, each preceded by a :. The following modifiers are defined:

h Removes a trailing pathname component

r Removes a trailing .xxx component

s/*l* /*r* /
    Substitutes *l* for *r*

t Removes all leading pathname components

& Repeats the previous substitution

g Applies the change globally, prefixing the above

p Prints the new command but does not execute it

q Quotes the substituted words, preventing substitutions

x Like q, but breaks into words at blanks, tabs, and newlines

Unless preceded by a g, the modification is applied only to the first modifiable word. In any case it is an error for no word to be applicable.

The left sides of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of /; a \ quotes the delimiter within the *l* and *r* strings. The character & in the right side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in !?*s* ?. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing ? in a contextual scan.

A history reference may be given without an event specification, e.g., !$. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus !?foo?^!$ gives the first and last arguments from the command matching ?foo?.

A special abbreviation of a history reference occurs when the first nonblank character of an input line is a ^. This is equivalent to !:s^, providing a convenient shorthand for substitutions on the text of the previous line. Thus ^lb^lib fixes the spelling of lib in the previous command. Finally, a history substitution may be surrounded with {

and } if necessary to insulate it from the characters that follow. Thus, after ls -ld ˜paul we might do !{l}a to do ls -ld ˜paula, while !la would look for a command starting la.

Quotations With ´ and "

The quotation of strings by ´ and " can be used to prevent all or some of the remaining substitutions. Strings enclosed in ´ are prevented any further interpretation. Variable and command expansion occurs in strings enclosed in ".

In both cases, the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitution* below) does a " quoted string yield parts of more than one word; ´ quoted strings never do.

Alias Substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for ls is ''ls -l'' the command ''ls /usr'' would map to ''ls -l /usr''. Similarly if the alias for ''lookup'' was ''grep \!^ /etc/passwd'' then ''lookup bill'' would map to ''grep bill /etc/passwd''.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can alias print ''´pr \!* | lpr´'' to make a command that paginates its arguments to the lineprinter.

There are four *csh* aliases distributed. These are **pushd, popd, swapd,** and **flipd.** These aliases maintain a directory stack.

pushd *dir*
: Pushes the current directory onto the top of the directory stack, then changes to the directory *dir*.

popd
Changes to the directory at the top of the stack, then removes (pops) the top directory from the stack, and announces the current directory.

swapd
Swaps the top two directories on the stack. The directory on the top becomes the second to the top, and the second to the top directory becomes the top directory.

flipd
Flips between two directories, the current directory and the top directory on the stack. If you are currently in **dir1**,and **dir2** is on the top of the stack, when **flipd** is invoked, you change to **dir2** and **dir1** is replaced as the top directory on the stack. When **flipd** is again invoked, you change to **dir1** and **dir2** is again the top directory on the stack.

Variable Substitution

The shell maintains a set of variables, each of which has a list of zero or more words as its value. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the **-v** command line option.

Other operations treat variables numerically. The at-sign (@) command permits numeric calculations to be performed and the result assigned to a variable. However, variable values are always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed, keyed by dollar sign ($) characters. This expansion can be prevented by preceding the dollar sign with a backslash (\) except within double quotation marks (") where it *always* occurs, and within single quotation marks (´) where it *never* occurs. Strings quoted by back quotation marks (`) are interpreted later (see *Command substitution* below) so dollar sign substitution does not occur there until later, if at all. A dollar sign is passed unchanged if followed by a blank, tab, or end-of-line.

Input and output redirections are recognized before variable expansion, and are expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in double quotation marks or given the :q modifier, the results of variable substitution may eventually be subject to command and filename substitution. Within double quotation marks ("), a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variable's value separated by blanks. When the :q modifier is applied to a substitution, the variable expands to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following sequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}
Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters, digits, and underscores.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}
May be used to select only some of the words from the value of *name*. The selector is subjected to $ substitution and may consist of a single number or two numbers separated by a -. The first word of a variables value is numbered 1. If the first number of a range is omitted it defaults to 1. If the last member of a range is omitted it defaults to $#name. The selector * selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
Gives the number of words in the variable. This is useful for later use in a [selector].

$0 Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}
Equivalent to $argv[number].

$* Equivalent to $argv[*].

The modifiers :h, :t, :r, :q and :x may be applied to the substitutions above as may :gh, :gt and :gr. If braces { } appear in the command form then the modifiers must appear within the braces. Only one : modifier is allowed on each $ expansion.

The following substitutions may not be modified with : modifiers.

$?name
${?name}
Substitutes the string 1 if name is set, 0 if it is not.

$?0 Substitutes 1 if the current input filename is known, 0 if it is not.

$$ Substitutes the (decimal) process number of the (parent) shell.

Command and Filename Substitution

Command and filename substitution are applied selectively to the arguments of built-in commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command Substitution

Command substitution is indicated by a command enclosed in back quotation marks. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded. This text then replaces the original string. Within double quotation marks, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename Substitution

If a word contains any of the characters *, ?, [or { or begins with the character ~, then that word is a candidate for filename substitution, also known as globbing. This word is then regarded as a pattern, and is replaced with an alphabetically sorted list of filenames which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing filename, but it is not required for each pattern to match. Only the metacharacters *, ?, and [imply pattern matching. The characters ~ and { are more akin to abbreviations.

In matching filenames, the character . at the beginning of a filename or immediately following a /, as well as the character / must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence within square brackets [] matches any one of the characters enclosed. Within square brackets [], a pair of characters separated by - matches any character lexically between the two.

The character ~ at the beginning of a filename is used to refer to home directories. Standing alone, it expands to the invoker's home directory contained in the variable HOME. When followed by a name consisting of letters, digits and - characters the shell searches for a user with that name and substitutes their home directory; thus ~ken might expand to /usr/ken and ~ken/chmach to /usr/ken/chmach. If the character ~ is followed by a character other than a letter or /, or if it does not appear at the beginning of a word, it is left unchanged.

The metanotation a{b,c,d}e is a shorthand for abe ace ade. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. Thus ~source/s1/{oldls,ls}.c expands to /usr/source/s1/oldls.c /usr/source/s1/ls.c, whether or not these files exist, assuming that the home directory for source is /usr/source. Similarly ../{memo,*box} might expand to ../memo ../box ../mbox. (Note that memo was not sorted with the results of matching *box.) As a special case {, } and {} are passed unchanged. This construct can be nested.

Spelling Checker

If the local variable *cdspell* has been set, the shell checks spelling whenever you use *cd* to change directories. For example, if you change to a different directory using *cd* and misspell the directory name, the shell responds with an alternative spelling of an existing directory. Enter "y" and press RETURN (or just press RETURN) to change to the offered directory. If the offered spelling is incorrect, enter "n", then retype the command line. In this example the **csh**(C) response is boldfaced:

```
% cd /usr/spol/uucp
/usr/spool/uucp? y
ok
```

Input/Output

The standard input and standard output of a command may be redirected with the following syntax:

< name
: Opens file *name* (after variable, command and filename expansion) as the standard input.

<< word
: Reads the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting backslash, double, or single quotation mark, or a back quotation mark appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote $, \ and `. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resulting text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name
: The file *name* is used as standard output. If the file does not exist, then it is created; if the file exists, it is overwritten.

  If the variable *noclobber* is set, then an error results if the file already exists or if it is not a character special file (e.g., a terminal or **/dev/null**). This helps prevent accidental destruction of files. In this case, the ! forms can be used to suppress this check.

  The forms involving & route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as < input filenames are.

>> name
>>& name
>>! name
>>&! name
: Uses file *name* as standard output like > but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the ! forms is given. Otherwise similar to >.

If a command is run in the background (followed by &) then the default standard input for the command is the empty file /dev/null. Otherwise, the command receives the input and output parameters from its parent shell. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form |& rather than just |.

Expressions

A number of the built-in commands (to be described later) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit, if,* and *while* commands. The following operators are available:

```
|| && | ^ & == != <= >= < > << >>
+ - * / % ! ~ ( )
```

Here the precedence increases to the right, == and !=, <=, >=, <, and >, << and >>, + and -, * / and % being, in groups, at the same level. The == and != operators compare their arguments as strings, all others operate on numbers. Strings which begin with 0 are considered octal numbers. Null or missing arguments are considered 0. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word unless a word is adjacent to components of expressions which are syntactically significant to the parser (& | < > ( )), it should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in { and } and file enquiries of the form -*l* name where *l* is one of:

r	Read access
w	Write access
x	Execute access
e	Existence
o	Ownership
z	Zero size
f	Plain file
d	Directory

Command and filename expansion is applied to the specified name, then the result is tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. 0. Command executions succeed, returning true, i.e. 1, if the command exits with status 0, otherwise they fail, returning false, i.e. 0.

If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

Control Flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. Due to the implementation, some restrictions are placed on the word placement for the *foreach, switch,* and *while* statements, as well as the *if-then-else* form of the *if* statement. Please pay careful attention to these restrictions in the descriptions in the next section.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto commands will succeed on nonseekable inputs.)

Built-In Commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, then it is executed in a subshell.

alias
alias name
alias name wordlist
The first form prints all aliases. The second form prints the alias for *name*. The final form assigns the specified *wordlist* as the alias of *name; wordlist* is command and filename substitution is applied to wordlist. *Name* is not allowed to be *alias* or *unalias*

break
Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while* statement. The remaining commands on the current line are executed. Multilevel breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a *switch,* resuming after the *endsw.*

case label:
This is part of the *switch* statement discussed below.

cd
cd name
chdir
chdir name
Changes the shell's working directory to directory *name.* If no argument is given, it then changes to the home directory of the user. If *name* is not found as a subdirectory of the current directory (and does not begin with /, ./, or ../), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable

whose value begins with /, then this is tried to see if it is a directory.

If *cdspell* has been set, the shell runs a spelling check as follows. If the shell is reading its commands from a terminal, and the specified directory does not exist (or some component cannot be searched), spelling correction is applied to each component of *directory* in a search for the "correct" name. The shell then asks whether or not to try and change the directory to the corrected directory name; an answer of *n* means "no," and anything else is taken as "yes."

continue

Continues execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:

Labels the default case in a *switch* statement. The default should come after all *case* labels.

echo wordlist

The specified words are written to the shell's standard output. A \c causes the echo to complete without printing a newline. A \n in *wordlist* causes a newline to be printed. Otherwise the words are echoed, separated by spaces.

else
end
endif
endsw

See the description of the *foreach, if, switch,* and *while* statements below.

exec command

The specified command is executed in place of the current shell.

exit
exit(expr)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)
...
end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* name*(wordlist)* and *end* must appear alone on separate lines.)

The built-in command *continue* may be used to continue the loop prematurely and the built-in command *break* to terminate it prematurely. When this command is read from the terminal, the contents of the loop are read by prompting with ? until end is typed before any statements in the loop are executed.

glob wordlist
Like *echo* but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to apply filename expansion to a list of words.

goto word
Filename and command expansion is applied to the specified *word* to yield a string of the form *label:*. The shell rewinds its input as much as possible and searches for a line of the form label: possibly preceded by blanks or tabs. Execution continues after the specified line.

history
Displays the history event list.

if (expr) command
If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, and *command* is **not** executed.

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif
If the specified *expr* is true then the commands before the first *else* are executed; else if *expr2* is true then the commands after the second then and before the second *else* are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* (expr) then must appear alone on its input line or after an *else.)*

logout
Terminates a login shell. The only way to log out if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command

The first form sets the *nice* for this shell to 4. By default, commands run under C-Shell have a ''nice value'' of 0. The second form sets the *nice* to the given number. The final two forms run *command* at priority 4 and *number* respectively. The super-user may specify negative niceness by using ''nice -number ....'' The command is always executed in a subshell, and the restrictions placed on commands in simple *if* statements apply.

nohup
nohup command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. Unless the shell is running in the background, *nohup* has no effect. All processes running in the background with & are automatically *nohup*ed.

onintr
onintr -
onintr label

Controls the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form, *onintr*, causes all interrupts to be ignored. The final form causes the shell to execute a *goto label* when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running in the background, interrupts are ignored whether any form of *onintr* is present or not.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in.

repeat count command

The specified *command*, which is subject to the same restrictions as the *command* in the simple *if* statement above, is executed *count* times. I/O redirection occurs exactly once, even if *count* is 0.

set
set name
set name=word

set name[index]=word
set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *indexth* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist*. Command and filename expansion is applied in all cases.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value

Sets the value of the environment variable *name* to be *value*, which must be a single string. Two useful environment variables are TERM, the type of your terminal and SHELL, the shell you are using.

shift
shift variable

In the first form, the members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as a value. The second form performs the same function on the specified variable.

source name

The shell reads commands from *name*. *Source* commands may be nested, but if they are nested too deeply, the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands, including the *csh* process from which *source* was called. If *source* is called from the login shell, it is logged out. Input during *source* commands is never placed on the history list.

switch (string)
case str1:
  ...
  breaksw
...
default:
  ...
  breaksw
endsw

Command and filename substitution is applied to *string*. The each case label is successively matched against the result. Variable expansion is also applied to the case labels, so the file metacharacters *, ?, and [...] can be used. If none of the labels match before a default label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels, as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command

With no argument, a summary of CPU time used by this shell and its children is printed. If arguments are given, the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes. *command* has the same restrictions as the simple *if* statement described above.

umask
umask value

The file creation mask is displayed (no arguments) or set to the specified value (one argument). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others, or 022 giving read and execute access to users in the group and all other users.

unalias pattern

All aliases whose names match the specified pattern are discarded. Thus, all aliases are removed by unalias *. It is not an error for nothing to be *unaliased.*

unhash

Use of the internal hash table to speed location of executed programs is disabled.

unset pattern

All variables whose names match the specified pattern are removed. Thus, all variables are removed by unset *; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset.*

wait

All child processes are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and process numbers of all children known to be outstanding.

while (expr)
...
end

While the specified expression evaluates nonzero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while*(expr) and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@
@ name = expr
@ name[index] = expr

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains <, >, & or | then at least this part of the expression must be placed within ( ). The third form assigns the value of *expr* to the *index*th argument of *name*. Both *name* and its *index*th component must already exist.

The operators *=, +=, etc. are available as in C. The space separating the name from the assignment operator is optional. Spaces are mandatory in separating components of *expr* which would otherwise be single words. The space between @ and *name* is also mandatory.

Special postfix ++ and -- operators increment and decrement *name* respectively, i.e. @ i++.

Predefined Variables

The following variables have special meaning to the shell. Of these, *argv, child, home, path, prompt, shell* and *status* are always set by the shell. Except for *child* and *status* this setting occurs only at initialization; these variables will not be modified unless done explicitly by the user.

The shell copies the environment variable PATH into the variable *path*, and copies the value back into the environment whenever *path* is set. Thus it is not necessary to worry about its setting other than in the file *.cshrc* since inferior *csh* processes will import the definition of *path* from the environment.

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e., $1 is replaced by $argv[1], etc. argv[0] is not defined, but $0 is.

cdpath Gives a list of alternate directories searched to find subdirectories in *cd* commands.

child The process number of the last command forked with &. This variable is *unset* when this process terminates.

echo

Set when the **-x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For nonbuilt-in commands all expansions occur before echoing. Built-in commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars

Can be assigned a two-character string. The first character is used as a history character in place of !, the second character is used in place of the ^ substitution mechanism. For example, set histchars=",;" will cause the history characters to be comma and semicolon.

history

Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. A *history* that is too large may run the shell out of memory. The last executed command is always saved on the history list.

home

The home directory of the invoker, initialized from the environment. The filename expansion of ~ refers to this variable.

ignoreeof

If set, the shell ignores end-of-file from input devices that are terminals. This prevents a shell from accidentally being terminated by pressing Ctrl-D.

mail

The files where the shell checks for mail. This check is executed after each command completion. The shell responds with, "You have new mail" if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric, it specifies a different mail checking interval: in seconds, rather than the default, which is 10 minutes.

If multiple mail files are specified, then the shell responds with "New mail in *name*", when there is mail in the file *name*.

noclobber

As described in the section *Input/Output,* restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that >> redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather, the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e., echo [still gives an error.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable, then only full pathnames will execute. The usual search path is /bin, /usr/bin, and ., but this may vary from system to system. For the super-user, the default search path is /etc, /bin and /usr/bin. A shell which is given neither the **-c** nor the **-t** option will normally hash the contents of the directories in the *path* variable after reading *.cshrc,* and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* command, or the commands may not be found.

prompt The string which is printed before reading each command from an interactive terminal input. If a ! appears in the string, it will be replaced by the current event number unless a preceding \ is given. Default is % , or # for the super-user.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Nonbuilt-In Command Execution* below.) Initialized to the home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Built-in commands which fail return exit status 1, otherwise these commands set status to 0.

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line to be sent to the screen displaying user time, system time, real time, and a utilization percentage which is the ratio of user plus system times to real time.

verbose Set by the **-v** command line option, causes the words of each command to be printed after history substitution.

Nonbuilt-In Command Execution

When a command to be executed is found to not be a built-in command, the shell attempts to execute the command via *exec*(S). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a **-c** nor a **-t** option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash),* or if the shell was given a **-c** or **-t** argument, and for each directory component of *path* which does not begin with a /, the shell concatenates each directory component of *path* with the given command name to form a pathname of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus

(cd ; pwd) ; pwd

prints the *home* directory; leaving you where you were and printing the name of the current directory, while

cd ; pwd

leaves you in the *home* directory. Parenthesized commands are always executed in a subshell. Thus

(cd; pwd); pwd

prints the home directory but leaves you in the original directory, while

cd; pwd

moves you to the home directory.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias are prepended to the argument list to form the shell command. The first word of the *alias* should be the full pathname of the shell (e.g. $shell). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument List Processing

If argument 0 to the shell is - then this is a login shell. The flag arguments are interpreted as follows:

-c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

-e The shell exits if any invoked command terminates abnormally or yields a nonzero exit status.

-f The shell will start faster, because it will neither search for nor execute commands from the file .cshrc in the invoker's home directory.

-i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their input and output are terminals.

-n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.

-v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

-x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

-V Causes the *verbose* variable to be set even before .cshrc is executed.

-X Causes the *echo* variable to be set even before .cshrc is executed.

After processing the flag arguments, if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options were given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by $0. On a typical system, most shell scripts are written for the standard shell (see *sh*(C)).

The C shell will execute such a standard shell if the first character of the script is not a # (i.e. if the script does not start with a comment). Remaining arguments initialize the variable *argv*.

Signal Handling

The shell normally ignores *quit* signals. The *interrupt* and *quit* signals are ignored for an invoked command if the command is followed by &; otherwise the signals have the values which the shell inherited from its parent. The shells handling of interrupts can be controlled by *onintr*. By default, login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file .logout.

Files

~/.cshrc	Read at by each shell at the beginning of execution
/etc/cshrc	Systemwide default *cshrc* file if none is present
~/.login	Read by login shell, after .cshrc at login
~/.logout	Read by login shell, at logout
/bin/sh	Shell for scripts not starting with a #
/tmp/sh*	Temporary file for <<
/dev/null	Source of empty file
/etc/passwd	Source of home directories for ~name

Limitations

Words can be no longer than 512 characters. The number of arguments to a command which involves filename expansion is limited to 1/6 the number of characters allowed in an argument list, which is 5120, less the characters in the environment. Also, command substitutions may substitute no more characters than are allowed in an argument list.

To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

See Also

access(S), exec(S), fork(S), pipe(S), signal(S), umask(S), wait(S), a.out(F), environ(M)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Built-in control structure commands like **foreach** and **while** cannot be used with |, & or ;.

Commands within loops, prompted for by ?, are not placed in the *history* list.

It is not possible to use the colon (:) modifiers on the output of command substitutions.

The C-shell has many built-in commands with the same name and functionality as Bourne shell commands. However, the syntax of these C-shell and Bourne shell commands often differs. Two examples are the *nice* and *echo* commands. Be sure to use the correct syntax when working with these built-in C-shell commands.

When a C-shell user logs in, the system reads and executes commands in */etc/cshrc* before executing commands in the user's *$HOME/.cshrc*. You can, therefore, modify the C-shell environment for all users on the system by editing */etc/cshrc*.

During intervals of heavy system load, pressing the delete key while at a C-shell prompt (%) may cause the shell to exit. If *csh* is the login shell, the user is logged out.

csh attempts to import and export the PATH variable for use with regular shell scripts. This only works for simple cases, where the PATH contains no command characters.

csplit

splits files according to context

Syntax

csplit [-s] [-k] [-**f** prefix] file arg1 [. . . argn]

Description

csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1*... *argn*. By default the sections are placed in files xx00 ... xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*.

01: From the line referenced by *arg1* up to the line referenced by *arg2*.

⋮

n+1: From the line referenced by *argn* to the end of *file*.

The options to *csplit* are:

-**s** *csplit* normally prints the character counts for each file created. If the -s option is present, *csplit* suppresses the printing of all character counts.

-**k** *csplit* normally removes created files if an error occurs. If the -**k** option is present, *csplit* leaves previously created files intact.

-**f** *prefix* If the -**f** option is used, the created files are named *prefix***00** ...*prefixn*. The default is **xx00** ... **xx***n*.

The arguments (*arg1* ... *argn*) to *csplit* can be a combination of the following:

/rexp/ A file is to be created for the section from the current line down to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or - some number of lines (e.g., **/Page/-5**).

%rexp% This argument is the same as */rexp/*, except that no file is created for the section.

lnno A file is to be created from the current line down to (but not including) *lnno*. The current line becomes *lnno*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotation marks. Regular expressions may not contain embedded newlines. *csplit* does not affect the original file; it is the users responsibility to remove it.

Examples

```
csplit -f cobol file '/procedure division/' '/par5./' '/par16./'
```

This example creates four files, **cobol00 ... cobol03**. After editing the ''split'' files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

See Also

ed(C), sh(C), regex(S)

Diagnostics

Self-explanatory except for:

```
arg - out of range
```

which means that the given argument did not reference a line between the current position and the end of the file.

Standards Conformance

csplit is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

cu

call another UNIX/XENIX system

Syntax

cu [**-s**speed] [**-l**line] [**-h**] [**-t**] [**-xn**] [**-o** | **-e** | **-oe**] [**-n**] telno
cu [**-s** speed] [**-h**] [**-xn**] [[**-o** | **-e** | **-oe**] **-l** line [**dir**]
cu [**-h**] [**-xn**] [**-o** | **-e** | **-oe**] systemname

Description

cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files.

cu accepts the following options and arguments:

-s*speed* Specifies the transmission speed (150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400). The default value is "Any" speed which will depend on the order of the lines in the **/usr/lib/uucp/Devices** file. A speed range can also be specified (for example, -s1200-4800).

-l*line* Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the **-l** option is used without the **-s** option, the speed of a line is taken from the Devices file. When the **-l** and **-s** options are both used together, cu will search the Devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/tty***ab*) in which case a telephone number (*telno*) is not required. The specified device need not be in the **/dev** directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* will not give the desired result (see *systemname* below).

-h Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

-t Used to dial an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

-x*n* Causes diagnostic traces to be printed; it produces a detailed output of the program execution on stderr. The debugging level, **n**, is a single digit; **-x9** is the most useful value.

-n For added security, will prompt the user to provide the telephone number to be dialed rather than taking it from the command line.

telno When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.

systemname A UUCP system name may be used rather than a telephone number. In this case, *cu* will obtain an appropriate direct line or telephone number from **/usr/lib/uucp/Systems.** Note: the *systemname* option should not be used in conjunction with the **-l** and **-s** options as *cu* will connect to the first available line for the system name specified, ignoring the requested line and speed.

dir The keyword **dir** can be used with **cu -l***line*, in order to talk directly to a modem on that line, instead of talking to another system via that modem. This can be useful when debugging or checking modem operation. Note: only users with write access to the *Devices* file are permitted to use **cu -l***line* **dir**.

In addition, *cu* uses the following options to determine communications settings:

-o If the remote system expects or sends 7-bit with odd parity.

-e If the remote system expects or sends 7-bit with even parity.

-oe
If the remote system expects or sends 7-bit, ignoring parity and sends 7-bit with either parity.

By default, *cu* expects and sends 8-bit characters without parity. If the login prompt received appears to contain incorrect 8-bit characters, or a correct login is rejected, use the 7-bit options described above.

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following user initiated commands:

~.	terminate the conversation.
~!	escape to an interactive shell on the local system.
~!*cmd*...	run *cmd* on the local system (via **sh -c**).
~$*cmd*...	run *cmd* locally and send its output to the remote system.
~%cd	change the directory on the local system. Note: **~!cd** will cause the command to be run by a sub-shell, probably not what was intended.
~%take *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%put *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places. For both **~%take** and **~%put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
~~*line*	send the line ~*line* to the remote system.
~%break	transmit a **BREAK** to the remote system (which can also be specified as **~%b**).
~%debug	toggles the **-x** debugging level between 0 and 9 (which can also be specified as **~%d**).
~t	prints the values of the termio structure variables for the user's terminal (useful for debugging).
~l	prints the values of the termio structure variables for the remote communication line (useful for debugging).

~%nostop toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with ~. Data from the remote is diverted (or appended, if >> is used) to *file* on the local system. The trailing ~> marks the end of the diversion.

The use of **~%put** requires *stty*(C) and *cat*(C) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of *echo*(S) and *cat*(C) on the remote system. Also, *tabs* mode (See *stty(C))* should be set on the remote system if tabs are to be copied without expansion to spaces.

When *cu* is used on *system1* to connect to *system2* and subsequently used on *system2* to connect to *system3*, commands on *system2* can be executed by using ~~. Executing a tilde command reminds the user of the local system *uname*. For example, *uname* can be executed on systems 1, 2, and 3 as follows:

```
uname
system3
~system1!uname
system1
~~system2!uname
system2
```

In general, ~ causes the command to be executed on the original machine, ~~ causes the command to be executed on the next machine in the chain.

Examples

To dial a system whose telephone number is 9 201 555 1212 using 1200 baud (where dialtone is expected after the 9):

```
cu -s1200 9=12015551212
```

If the speed is not specified, ''Any'' is the default value.

To login to a system connected by a direct line:

```
cu -l /dev/ttyXX
```

or

cu -l ttyXX

To dial a system with the specific line and a specific speed:

cu -s1200 -l ttyXX

To dial a system using a specific line associated with an auto dialer:

cu -l ttyXX 9=12015551212

To use a system name:

cu systemname

To talk directly to an ACU (connect directly with the modem and enter modem commands manually):

cu -lttyXX dir

Files

/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/lib/uucp/LCK..(tty-device)

See Also

cat(C), ct(C), echo(S), stty(C), uucp(C), uname(C)

Diagnostics

Exit code is zero for normal exit, otherwise, one.

Warnings

The *cu* command does not do any integrity checking on data it transfers. Data fields with special *cu* characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a ~. to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands. *cu* between an IMBR1 and a penril modem will not return a login prompt immediately upon connection. A carriage return will return the prompt.

Notes

There is an artificial slowing of transmission by *cu* during the **~%put** operation so that loss of data is unlikely.

Standards Conformance

cu is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

cut

cuts out selected fields of each line of a file

Syntax

cut -c list [file1 file2 ...]
cut -f list [**-d** char] [-s] [file1 file2 ...]

Description

Use *cut* to cut out columns from a table or fields from each line of a file. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (**-c** option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (**-f** option). *cut* can be used as a filter. If no files are given, the standard input is used.

The meanings of the options are:

list A comma-separated list of integers (in increasing order), with an optional dash (-), indicates ranges, as in the **-o** option of *nroff/troff* for page ranges; e.g., **1,4,7**; **1-3,8**; **-5,10** (short for **1-5,10**); or **3-** (short for third through last field).

-c*list* The *list* following **-c** (no space) specifies character positions (e.g., **-c1-72** would keep the first 72 characters of each line).

-f*list* The *list* following -f is a list of fields assumed to be separated in the file by a delimiter character (see **-d**); e.g., **-f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless **-s** is specified.

-d*char* The character following **-d** is the field delimiter (**-f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.

-s If the **-f** option is used, **-s** suppresses lines with no delimiter characters. Unless specified, lines with no delimiters will be passed through untouched.

Either the **-c** or **-f** option must be specified.

Notes

Use *grep*(C) to make horizontal ''cuts'' (by context) through a file, or *paste*(C) to put files together horizontally. To reorder columns in a table, use *cut* and *paste*.

Examples

cut -d: -f1,5 /etc/passwd	Maps user ID's to names.
name=\`who am i | cut -f1 -d" "\`	Sets *name* to current login name.

See Also

grep(C), paste(C)

Diagnostics

line too long A line can have no more than 511 characters or fields.

bad list for c / f option
Missing **-c** or **-f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

no fields The *list* is empty.

Standards Conformance

cut is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

date

prints and sets the date

Syntax

date [-**cms**] [mmddhhmm[yy]] [+format]

Description

If no argument is given, or if the argument begins with +, the current date and time are printed as defined by the locale. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24-hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

date 10080045

sets the date to Oct 8, 12:45 AM, if the local language is set to English. The current year is the default if no year is mentioned. The system operates in GMT. *date* takes care of the conversion to and from local standard and daylight time.

If the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf* (S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by a percent sign (%) and will be replaced in the output by its corresponding value. A single percent sign is encoded by doubling the percent sign, i.e., by specifying "%%". All other characters are copied to the output without change. The string is always terminated with a newline character.

Field Descriptors:

n Inserts a newline character

t Inserts a tab character

m Month of year - 01 to 12

d Day of month - 01 to 31

y Last 2 digits of year - 00 to 99

D Date as mm/dd/yy

H Hour - 00 to 23

M Minute - 00 to 59

S Second - 00 to 59

T Time as HH:MM:SS

j Julian date - 001 to 366

w Day of the week - Sunday = 0

a Abbreviated weekday - Sun to Sat

h Abbreviated month - Jan to Dec

r Time in AM/PM notation

Options

-c Prints the current date and time from the hardware real-time clock. Thus, **date -c** *mmddhhmm*[*yy*] sets the real-time clock.

-m Updates the year on the hardware real-time clock, if it is January 1, and makes adjustments to the real-time clock if it is February 29 in a leap year. These dates are not automatically incremented. Be sure to use this option after midnight. The **-m** option determines if it is January 1 or February 29, and then updates the hardware real-time clock if necessary. For the **-m** option to work correctly, the software clock and the hardware clock should be within twelve hours of each other. Use *cron*(C) to execute **date m** each day.

-s Sets (synchronizes) the system (i.e., software) clock to the current time and date from the hardware real-time clock.

The operating system normally uses only the system (software) clock. It uses the hardware real-time clock only with the *date* command.

Example

The line

```
date ´+DATE: %m/%d/%y%nTIME: %H:%M:%S´
```

generates as output:

```
DATE: 08/01/90
TIME: 14:45:05
```

Diagnostics

no permission	You aren't the super-user and you are trying to change the date.
bad conversion	The date set is syntactically incorrect.
bad format character	The field descriptor is not recognizable.

Standards Conformance

date is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

dc

invokes an arbitrary precision calculator

Syntax

dc [file]

Description

dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but you may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number
: The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ - / * % ^
: The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack and the result pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x*
: The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x*
: The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d
: The top value on the stack is duplicated.

p
: The top value on the stack is printed. The top value remains unchanged. **p** interprets the top of the stack as an ASCII string, removes it, and prints it.

f
: All values on the stack are printed.

q Exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x Treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X Replaces the number on the top of the stack with its scale factor.

[...] Puts the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x*
The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! Interprets the rest of the line as an Altos UNIX System V command.

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input.

I Pushes the input base on the top of the stack.

o The top value on the stack is popped and used as the number radix for further output.

O Pushes the output base on the top of the stack.

k The top of the stack is popped, and that value is used as a non-negative scale factor; the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

Z Replaces the number on the top of the stack with its length.

? A line of input is taken from the input source (usually the terminal) and executed.

; : Used by *bc* for array operations.

Example

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

See Also

bc(C)

Diagnostics

x is unimplemented	The octal number *x* corresponds to a character that is not implemented as a command
stack empty	Not enough elements on the stack to do what was asked
Out of space	The free list is exhausted (too many digits)
Out of headers	Too many numbers being kept around
Out of pushdown	Too many items on the stack
Nesting Depth	Too many levels of nested execution

Notes

bc is a preprocessor for *dc*, providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs. For interactive use, *bc* is preferred to *dc* .

dd

converts and copies a file

Syntax

dd [option=value] ...

Description

dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

Option	*Value*
if=*file*	Input filename; standard input is default
of=*file*	Output filename; standard output is default
ibs=*n*	Input block size is *n* bytes (default is BSIZE block size)
obs=*n*	Output block size (default is BSIZE block size)
bs=*n*	Sets both input and output block size, superseding *ibs* and *obs*. If no conversion is specified, it is particularly efficient since no in-core copy needs to be done
cbs=*n*	Conversion buffer size
skip=*n*	Skips *n* input records before starting copy
seek=*n*	Seeks *n* records from beginning of output file before copying
count=*n*	Copies only *n* input records
conv=ascii	Converts EBCDIC to ASCII
conv=ebcdic	Converts ASCII to EBCDIC
conv=ibm	Slightly different map of ASCII to EBCDIC
conv=lcase	Maps alphabetic characters to lowercase

Option	*Value*
conv=ucase	Maps alphabetic characters to uppercase
conv=swab	Swaps every pair of bytes
conv=sync	Pads every input record to *ibs*
conv="...,..."	Several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and newline added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output record of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

Examples

This command reads an EBCDIC tape, blocked ten 80-byte EBCDIC card images per record, into the ASCII file **outfile** :

```
dd if=/dev/rmt0 of=outfile ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *dd* is especially suited to I/O on raw physical devices because it allows reading and writing in arbitrary record sizes.

See Also

copy(C), cp(C), tar(C)

Diagnostics

f+p records in(out)	Numbers of full and partial records read(written)

Notes

The ASCII/EBCDIC conversion tables are taken from the 256-character standard in the CACM Nov, 1968. The *ibm* conversion corresponds better to certain IBM print train conventions. There is no universal solution.

Newlines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC.

When using *dd* with a raw device, specify the block size as a multiple of 512-byte blocks. For example, to use a 9K block size, enter:

```
dd if=file of=/dev/rfd0 bs=18b
```

You could also enter:

```
dd if=file of=/dev/rfd0 bs=9K
```

Standards Conformance

dd is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

devnm

identifies device name

Syntax

/etc/devnm [names]

Description

devnm identifies the special file associated with the mounted file system where the argument *name* resides.

This command is most commonly used by the **/etc/rc2** scripts to construct a mount table entry for the **root** device.

Examples

Be sure to type full pathnames in this example:

/etc/devnm /usr

If **/dev/hdb** is mounted on **/usr**, this produces:

hdb /usr

Files

/dev/*	Device names
/etc/rc2	Startup commands

See Also

setmnt(ADM)

Standards Conformance

devnm is conformant with:
AT&T SVID Issue 2, Select Code 307-127.

df

report number of free disk blocks

Syntax

df [**-t**] [**-f**] [-v] [filesystems]

Description

df prints out the number of free blocks and free inodes available for on-line filesystems by examining the counts kept in the super-blocks; *filesystems* may be specified by device name (e.g., **/dev/root**). If the *filesystems* argument is unspecified, the free space on all of the mounted filesystems is sent to the standard output. The list of mounted filesystems is given in **/etc/mnttab**.

Options include:

-t Causes total allocated block figures to be reported as well as number of free blocks.

-f Reports only an actual count of the blocks in the free list (free inodes are not reported). With this option, df reports on raw devices.

-v Reports the percent of blocks used as well as the number of blocks used and free.

The **-v** option can not be used with other *df* options.

Files

/dev/*
/etc/mnttab

See Also

mount(ADM), fsck(ADM), mnttab(F)

Notes

See *Notes* under *mount*(ADM).

This utility reports sizes in 512 byte blocks. *df* will report 2 blocks less free space, rather than 1 block, since the file uses one system block of 1024 bytes.

The directory **/etc/fscmd.d/***TYPE* contains programs for each filesystem type, *df* invokes the appropriate binary.

Authorization

The behavior of this utility is affected by assignment of the **queryspace** authorization, which is usually reserved for system administrators. Refer to the ''Using a Trusted System'' chapter of the *User's Guide* for more details.

Standards Conformance

df is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

diff

compares two text files

Syntax

diff [**-befh**] file1 file2

Description

diff tells what lines must be changed in two files to bring them into agreement. If *file1* or *file2* is a dash (-), the standard input is used. If *file1* or *file2* is a directory, *diff* uses the file in that directory that has the same name as the file (*file2* or *file1* respectively) it is compared to. For example:

diff */tmp dog*

compares the file named *dog* that is in the */tmp* directory, with the file *dog* in the current directory. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward, one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where $n1 = n2$ or $n3 = n4$ are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

The **-b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **-e** option produces a script of *a, c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **-f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **-e**, the following shell procedure helps maintain multiple versions of a file:

(shift; cat $*; echo ´1,$p´) | ed - $1

This works by performing a set of editing operations on an original ancestral file. This is done by combining the sequence of *ed* scripts given as all command line arguments except the first. These scripts are presumed to have been created with *diff* in the order given on the command line. The set of editing operations is then piped as an editing script to *ed* where all editing operations are performed on the ancestral file given as the first argument on the command line. The final version of the file is then printed on the standard output. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand.

Except in rare circumstances, *diff* finds the smallest sufficient set of file differences.

The **-h** option does a fast, less-rigorous job. It works only when changed stretches are short and well separated, but the files can be of unlimited length. The **-e** and **-f** options cannot be used with the **-h** option.

Files

/tmp/d?????

/usr/lib/diffh (executable used when **-h** option is specified)

See Also

cmp(C), comm(C), ed(C)

Diagnostics

Exit status is 0 for no differences, 1 for some differences, 2 for errors.

Notes

Editing scripts produced under the **-e** or **-f** option do not always work correctly on lines consisting of a single period (.).

Standards Conformance

diff is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

diff3

compares three files

Syntax

diff3 [**-ex3**] file1 file2 file3

Description

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	All three files differ
====1	*File1* is different
====2	*File2* is different
====3	*File3* is different

The type of change suffered in converting a given range of a given file to some other range is indicated in one of these ways:

f : *n1* **a**	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f : *n1* , *n2* **c**	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the **-e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3. The **-x** option produces a script to incorporate changes flagged with "====". Similarly, the **-3** option produces a script to incorporate changes flagged with "====3". The following command applies a resulting editing script to *file1* :

```
(cat script; echo '1,$p') | ed - file1
```

Files

/tmp/d3*

/usr/lib/diff3prog

See Also

diff(C)

Notes

The **-e** option does not work properly for lines consisting of a single period.

The input file size limit is 64K bytes.

dircmp

compares directories

Syntax

dircmp [**-d**] [**-s**] [**-w***n*] dir1 dir2

Description

dircmp examines *dir1* and *dir2* and generates tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated in addition to a list that indicates whether the files common to both directories have the same contents.

There are three options available:

-d Performs a full *diff* on each pair of like-named files if the contents of the files are not identical.

-s Suppresses output of identical filenames.

-w*n* Changes the width of the output line to *n* characters. The default width is 72.

See Also

cmp(C), diff(C)

Standards Conformance

dircmp is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

dirname

delivers directory part of pathname

Syntax

dirname string

Description

dirname delivers all but the last component of the pathname in *string* and prints the result on the standard output. If there is only one component in the pathname, only a "dot" is printed. It is normally used inside substitution marks (\` \`) within shell procedures.

The companion command *basename* deletes any prefix ending in a slash (/) and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output.

Examples

The following example sets the shell variable NAME to **/usr/src/cmd**:

```
NAME=`dirname /usr/src/cmd/cat.c`
```

This example prints **/a/b/c** on the standard output:

```
dirname /a/b/c/d
```

This example prints a "dot" on the standard output:

```
dirname file.ext
```

This example moves to the location of a file being searched for (*lostfile*):

cd 'find . -name *lostfile* -exec dirname { } ;'

See Also

basename(C), sh(C)

Standards Conformance

dirname is conformant with:

The X/Open Portability Guide II of January 1987.

disable

turns off terminals and printers

Syntax

disable tty ...
disable [**-c**][**-r**[reason]] printers

Description

For terminals, this program manipulates the **/etc/conf/cf.d/init.base** file and signals *init* to disallow logins on a particular terminal. For printers, *disable* stops print requests from being sent to the named printer. The following options can be used:

-c Cancels any requests that are currently printing.

-r[*reason*] Associates a *reason* with disabling the printer. The *reason* applies to all printers listed up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a *reason*, then a default *reason* is used. *Reason* is reported by *lpstat*(C).

Examples

In this example, a printer named *linepr* is disabled because of a paper jam:

disable -r"paper jam" linepr

Files

/dev/tty*

/etc/conf/cf.d/init.base

/usr/spool/lp/*

See Also

login(M), enable(C), inittab(F), getty(M), init(M), lp(C), lpstat(C), uugetty(ADM)

Authorization

The behavior of this utility is affected by assignment of the **printerstat** authorization, which is usually reserved for system administrators. Refer to the "Using a Trusted System" chapter of the *User's Guide* for more details.

diskcp, diskcmp

copies or compares floppy disks

Syntax

diskcp [-f] [-d] [-s] [-48ds9] [-96ds9] [-96ds15] [-135ds9] [-135ds18]
diskcmp [-d] [-s] [-48ds9] [-96ds9] [-96ds15] [-135ds9] [-135ds18]

Description

diskcp is used to make an image (exact copy) of a source floppy disk on a target floppy disk. On machines with one floppy drive *diskcp* temporarily transfers the image to the hard disk until a ''target'' floppy is inserted into the floppy drive. On machines with two floppy drives *diskcp* immediately places the image of the source floppy directly on the target floppy.

diskcmp functions similarly to *diskcp*. It compares the contents of one floppy disk with the contents of a second floppy disk using the *cmp* utility.

The options are:

-f Format the target floppy disk before the image is copied (*diskcp* only).

-d The computer has dual floppy drives. *diskcp* copies the image directly onto the target floppy.

-s Uses *sum*(C) to compare the contents of the source and target floppies; gives an error message if the two do not match.

-48ds9
This setting is for low density 48tpi floppies. It is the default setting.

-96ds9
This setting is for high density 96tpi floppies.

-96ds15
This setting is for quad density 96tpi floppies.

-135ds9
This setting is for high density 135tpi 3.5 inch floppies.

-135ds18
This setting is for quad density 135tpi 3.5 inch floppies.

When using the -96ds9 and -96ds15 options of *diskcp* without the **-f** option, if the first target disk is unformatted, the program will note it, format it and make the copy. If another copy is requested and another unformatted target disk inserted, **diskcp** exits with a "System error." Quit, format the floppy, and reinvoke **diskcp** to make another copy.

Examples

To make a copy of a floppy, place the source floppy in the drive and type:

diskcp

When *diskcp* is finished copying to the hard disk, it prompts you to insert the target floppy in the drive. If you specify the **-f** flag when you invoke *diskcp* , the program formats the target floppy. When the copy is finished, *diskcp* asks if you would like to make another copy of the same source disk. If you enter 'n', it asks if you would like to copy another source disk.

Specify the **-d** flag on the command line if you have two floppy drives:

diskcp -d

Notes

If *diskcp* encounters a write error while copying the source image to the target disk, it formats the disk and tries to write the source image again. This happens most often when an unformatted floppy is used and the **-f** flag is not specified.

Files

/usr/bin/diskcp
/usr/bin/diskcmp
/tmp/disk????

See Also

cmp(C), dd(C), format(C), sum(C)

Value Added

diskcmp and *diskcp* are extensions of AT&T System V provided by Altos UNIX System V.

dos: doscat, doscp, dosdir, dosformat, dosmkdir, dosls, dosrm, dosrmdir

access to and manipulation of DOS files and DOS filesystems

Syntax

doscat [-r | -m] file ...

doscp [-r | -m] file1 file2

doscp [-r | -m] file ... directory

dosdir directory ...

dosformat [-fqv] drive

dosls directory ...

dosmkdir directory ...

dosrm file ...

dosrmdir directory ...

Description

The *dos* commands provide access to the files and directories on MS-DOS disks and on a DOS partition of a hard disk. Note that in order to use these commands on a DOS partition of a hard disk, the partition must be bootable, although not active. It is also possible to mount and access a DOS filesystem while operating from the Altos UNIX System V partition.

The *dos* commands perform the following actions:

doscat — Copies one or more DOS files to the standard output. If **-r** is given, the files are copied without newline conversions. If **-m** is given, the files are copied with newline conversions (see "Conversions" below).

doscp — Copies files between a DOS disk and an Altos UNIX System V filesystem. If *file1* and *file2* are given, *file1* is copied to *file2*. If a *directory* is given, one or more files are copied to that directory. If **-r** is given, the files are copied without newline conversions. If **-m** is given, the files are copied with newline conversions (see "Conversions" below).

dosdir Lists DOS files in the standard DOS style directory format.

dosformat Creates a DOS 2.0 formatted diskette. The drive may be specified in either DOS drive convention, using the default file **/etc/default/msdos**, or using the Altos UNIX System V special file name. *dosformat* cannot be used to format a hard disk. The **-f** option suppresses the interactive feature. The **-q** (quiet) option is used to suppress information normally displayed during *dosformat* . The **-q** option does not suppress the interactive feature. The **-v** option prompts the user for a volume label after the diskette has been formatted. The maximum size of the volume label is 11 characters.

dosls Lists DOS directories and files in an Altos UNIX System V format (see *ls*(C)).

dosrm Removes files from a DOS disk.

dosmkdir Creates a directory on a DOS disk.

dosrmdir Deletes directories from a DOS disk.

The *file* and *directory* arguments for DOS files and directories have the form:

device:name

where *device* is an Altos UNIX System V pathname for the special device file containing the DOS disk, and *name* is a pathname to a file or directory on the DOS disk. The two components are separated by a colon (:). For example, the argument:

/dev/fd0:/src/file.asm

specifies the DOS file, **file.asm**, in the directory, **/src**, on the disk in the device file **/dev/fd0**. Note that slashes (and not backslashes) are used as filename separators for DOS pathnames. Arguments without a *device:* are assumed to be Altos UNIX System V files.

For convenience, the user configurable default file, **/etc/default/msdos**, can define DOS drive names to be used in place of the special device file pathnames. It can contain lines with the following format:

A=/dev/fd0
C=/dev/hdad
D=/dev/hdbd

The drive letter "A" may be used in place of special device file pathname **/dev/fd0** when referencing DOS files (see "Examples" below). The drive letter "C" or "D" refer to the DOS partition on the first or second hard disk.

The commands operate on the following kinds of disks:

DOS partitions on a hard disk
5 1/4 inch DOS
3 1/2 inch DOS
8, 9, 15, or 18 sectors per track
40 tracks per side
1 or 2 sides
DOS versions 1.0, 2.0 or 3.0

Conversions

In the case of *doscp*, certain conversions are performed when copying an Altos UNIX System V file. Filenames with a basename longer than eight characters are truncated. Filename extensions (the part of the name following separating period) longer than three characters are truncated. For example, the file 123456789.12345 becomes 12345678.123. A message informs the user that the name has been changed and the altered name is displayed. Filenames containing illegal DOS characters are stripped when writing to the MS-DOS format. A message informs the user that characters have been removed and displays the name as written.

All DOS text files use a carriage-return/linefeed combination, CR-LF , to indicate a newline. Altos UNIX System V files use a single newline LF character. When the *doscat* and *doscp* commands transfer DOS text files to the Altos UNIX System V filesystem, they automatically strip the CR. When text files are transferred to DOS , the commands insert a CR before each LF character.

Under some circumstances the automatic newline conversions do not occur. The **-m** option may be used to ensure the newline conversion. The **-r** option can be used to override the automatic conversion and force the command to perform a true byte copy regardless of file type.

Examples

```
doscat /dev/fd0:/docs/memo.txt
doscat /tmp/f1 /tmp/f2 /dev/fd0:/src/file.asm
```

```
dosdir /dev/fd0:/src
dosdir A:/src A:/dev

doscp /tmp/myfile.txt /dev/fd0:/docs/memo.txt
doscp /tmp/f1 /tmp/f2 /dev/fd0:/mydir
dosformat /dev/fd0

dosls /dev/fd0:/src
dosls B:

dosmkdir /dev/fd0:/usr/docs

dosrm /dev/fd0:/docs/memo.txt
dosrm A:/docs/memo1.txt

dosrmdir /dev/fd0:/usr/docs
```

Accessing DOS Filesystems From the UNIX Partition

The ability to mount DOS filesystems is an extension of the DOS utilities documented here.

There are several limitations with the DOS directory structure which makes this a difficult task. These limitations are due to insufficient information when compared to the Altos UNIX System V filesystem.

The DOS directory structure contains the following information:

- *Filename:* up to 8 characters with 3 character extension (**foo.bat**)
- *File Attribute:* read-only/read-write, hidden/visible file, system/normal file, Volume name/normal file name, subdirectory/normal file, archive/modified bit
- *Time of last modification*
- *Date of last modification*
- *Starting point* (reference through FAT)
- *File size in bytes*

Using this information, it is converted to an actual UNIX inode. There are some Altos UNIX System V provisions that cannot be carried over, because the filesystem must remain sane under DOS.

- Any date in the UNIX inode table for the DOS filesystem is the same as the modification date (ctime = atime = mtime).
- The only types of nodes allowed in the DOS filesystem are

directories and normal files. Pipes, semaphores, and special device files do not exist because they do not have a counterpart under DOS.

- The permissions are 0777 for readable/writable files and 0555 for read only files. If a user can access the filesystem, the user will be limited by the permissions available under the DOS directory structure. This permission is read-only or read write. When creating a file, the creator's umask/mode is examined. The creation mode is based on the owner write bit.

- The gid/uid for all files on the DOS filesystem is the same as the mountpoint. The mount point will maintain the necessary security. If a user can get into the mountpoint, then the user has the same access as the owner.

- There is only one link for each file under the DOS filesystem. "." and ".." are a special case and are not links.

- On every change of the modification time (which on an Altos UNIX System V system would change atime, ctime, mtime) the DOS archive bit is set.

- Following DOS filesystem requirements, all blocks previous to a written block are allocated before the original block is written. This differs from Altos UNIX System V systems where the program may seek out beyond the end of a file and write a block. Altos UNIX System V systems do not necessarily write blocks that have been skipped over.

- If a program does not use the *directory*(S) system calls, but opens the directory in the DOS filesystem as a file, the program should see the DOS directory structure as it really exists. By using the *directory*(S) system calls, the filesystem switch code will put together an Altos UNIX System V style directory entry.

- File contents are not mapped from the DOS filesystem. The file appears exactly as it is under DOS. For example, \r\n combinations are left as \r\n and not mapped to just \n. The file and directory names are mapped to uppercase.

DOS File Conversion

The utilities *xtod*(C) and *dtox*(C) can be used to convert the EOL sequences used to and from DOS, respectively.

Files

/etc/default/msdos	Default information
/dev/fd*	Floppy disk devices
/dev/hd*	Hard disk devices

See Also

assign(C), dtype(C), mkfs(ADM), dtox(C), xtod(C), and "Using DOS" in the *System Administrator's Guide*

Notes

Using the DOS utilities, is not possible to refer to DOS directories with wild card specifications. The programs mentioned above cooperate among themselves so no two programs will access the same DOS disk. Only one process will access a given DOS disk at any time, while other processes wait. If a process has to wait too long, it displays the error message, "can't seize a device," and exits with an exit code of 1.

You cannot use the *dosformat* command to format device A: because it is aliased to **/dev/install**, which cannot be formatted. Use **/dev/rfd0/** instead.

The following hard disk devices:

/dev/hdad
/dev/rhdad
/dev/hdbd
/dev/rhdbd

are similar to **/dev/hdaa** in that the disk driver determines which partition is the DOS partition and uses that as *hd?d*. This means that software using the DOS partition does not need to know which partition is DOS.

The Development System supports the creation of DOS executable files, using *cc* (CP). Refer to the *C User's Guide* and *C Library Guide* for more information on using your Altos UNIX System V system to create programs suitable for DOS systems.

All of the DOS utilities leave temporary files in */tmp*. These files are automatically removed when the system is rebooted. They can also be manually removed.

You must have DOS 3.3 or earlier. Extended DOS partitions are not supported.

Value Added

doscat, *doscp*, *dosdir*, *dosformat*, *dosls*, *dosmkdir*, *dosrm* and *dosrmdir* are extensions of AT&T System V provided by Altos UNIX System V.

dtox

change file format from MS-DOS to UNIX

Syntax

dtox *filename* > *output.file*

Description

The *dtox* command converts a file from MS-DOS format to Altos UNIX System V format. MS-DOS files terminate a line of text with a carriage return and a linefeed, while Altos UNIX System V files terminate a line with a linefeed only. Also MS-DOS places a ⟨CTL⟩z at the end of a file, while Altos UNIX System V systems do not. Some programs and utilities are sensitive to this difference and some are not. If a text or data file is not being interpreted correctly, then use the *dtox* and *xtod* conversion utilities. The *dtox* command strips the extra carriage return from the end of each line and strips the ⟨CTL⟩z from the end of the file. This utility is not required for binary object files.

If no filename is specified on the command line, *dtox* takes input from standard input. Output of the utility goes to standard output.

See Also

xtod(C)

Value Added

dtox is an extension of AT&T System V provided by Altos UNIX System V.

dtype

determines disk type

Syntax

dtype [-s] device ...

Description

dtype determines type of disk, prints pertinent information on the standard output unless the silent (-s) option is selected, and exits with a corresponding code (see below). When more than one argument is given, the exit code corresponds to the last argument.

Disk Type	Exit Code	Message (optional)
Misc.	60	error (specified)
	61	empty or unrecognized data
Storage	70	*backup* format, volume n
	71	*tar* format[, extent e of n]
	72	*cpio* format
	73	*cpio* character (-c) format
MS-DOS	80	DOS 1.x, 8 sec/track, single sided
	81	DOS 1.x, 8 sec/track, dual sided
	90	DOS 2.x, 8 sec/track, single sided
	91	DOS 2.x, 8 sec/track, dual sided
	92	DOS 2.x, 9 sec/track, single sided
	93	DOS 2.x, 9 sec/track, dual sided
	94	DOS 2.x, fixed disk
	110	DOS 3.x, 9 sec/track, dual sided
XENIX	120	XENIX 2.x filesystem [needs cleaning]
	130	XENIX 3.x or later filesystem [needs cleaning]
UNIX	140	UNIX 1K filesystem [needs cleaning]

Notes

word-swapped refers to byte ordering of long words in relation to the host system.

XENIX filesystems and *backup* and *cpio* binary formats may not be recognized if created on a foreign system. This is due to such system differences as byte and word swapping and structure alignment.

This utility only works reliably for floppy diskettes.

Value Added

dtype is an extension of AT&T System V provided by Altos UNIX System V.

du

summarizes disk usage

Syntax

du [**-afrsu**] [names]

Description

du gives the number of blocks contained in all files and directories recursively within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, the current directory is used.

-s causes only the grand total (for each of the specified *names*) to be given. **-a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

The **-f** option causes **du** to display the usage of files in the current file system only. Directories containing mounted file systems will be ignored. The **-u** option causes *du* to ignore files that have more than one link.

du is normally silent about directories that cannot be read, files that cannot be opened, etc. The **-r** option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

Notes

If the **-a** option is not used, nondirectories given as arguments are not listed.

If there are too many distinct linked files, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

This utility reports sizes in 512 byte blocks. *du* interprets 1 block from a 1024 byte block system as 2 of its own 512 byte blocks. Thus a 500 byte file is interpreted as 2 blocks rather than 1.

Standards Conformance

du is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

echo

echo

Syntax

echo [-n] [arg] ...

Description

The *echo* command writes its arguments separated by blanks and terminated by a new-line on the standard output. The **-n** option prints a line without the new-line; same as using the **\c** escape sequence.

echo also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b backspace
\c print line without new-line
\f form-feed
\n new-line
\r carriage return
\t tab
\v vertical tab
**** backslash
*****n* The 8-bit character whose ASCII code is a 1, 2 or 3-digit octal number. In all cases, *n* must start with a zero. For example:

echo "\07 "	- Echoes Ctrl-G.
echo "\007 "	- Also echoes Ctrl-G.
echo "\065 "	- Echoes the number "5".
echo "\0101 "	- Echoes the letter "A".

The *echo* command is useful for producing diagnostics in command files and for sending known data into a pipe.

See Also

sh(C)

Notes

When representing an 8-bit character by using the escape convention **\0***n*, the *n* must always be preceded by the digit zero (0).

For example, typing: **echo 'WARNING:\07'** will print the phrase **WARNING:** and sound the "bell" on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the "\" that precedes the "07".

For the octal equivalents of each character, see *ascii*(M).

ed, red

invokes the text editor

Syntax

ed [-] [**-p** string] [file]

red [file]

Description

ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. *ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

red is a restricted version of *ed*(C). It will only allow editing of files in the current directory. It prohibits executing *sh*(C) commands via the ! command. *red* displays an error message on any attempt to bypass these restrictions.

In general, *red* does not allow commands like

    !date

or

    !sh

Furthermore, *red* will not allow pathnames in its command line. For example, the command:

    red /etc/passwd

when the current directory is not /etc causes an error.

Options

The options to *ed* are:

- Suppresses the printing of character counts by the *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and the ! prompt after a !*shell command.*

-p Allows the user to specify a prompt string.

ed supports formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty -tabs or stty tab3** mode (see *stty*(C)), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: While inputing text, tab characters are expanded to every eighth column as the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by entering a period (.) alone at the beginning of a line.

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression specifies a set of character strings. A member of this set of strings is said to be *matched* by the regular expression. The regular expressions allowed by *ed* are constructed as follows:

The following one-character regular expressions match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash (\) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

a. ., *, [, and \ (dot, star, left square bracket, and backslash, respectively), which are otherwise special, *except* when they appear within square brackets ([ ]); see 1.4 below).

b. ^ (caret), which is special at the *beginning* of an *entire* regular expression (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([ ]) (see 1.4 below).

c. **$** (dollar sign), which is special at the *end* of an entire regular expression (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire regular expression, which is special for that regular expression (for example, see how slash (/) is used in the *g* command below).

1.3 A period (.) is a one-character regular expression that matches any character except newline.

1.4 A nonempty string of characters enclosed in square brackets ([ ]) is a one-character regular expression that matches *any one* character in that string. If, however, the first character of the string is a caret (^), the one-character regular expression matches any character *except* newline and the remaining characters in the string. The star (*) also has this special meaning *only* if it occurs first in the string. The dash (-) may be used to indicate a range of consecutive ASCII characters; for example, [0-9] is equivalent to [0123456789]. The dash (-) loses this special meaning if it occurs first (after an initial caret (^), if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial caret (^), if any); e.g., [ ]a-f] matches either a right square bracket (]) or one of the letters "a" through "f" inclusive. Dot, star, left bracket, and the backslash lose their special meaning within such a string of characters.

Ranges of characters (characters separated by -) are treated according to the current locale's collation sequence (see *locale*(M)). Therefore, if the collation sequence in use is A, a, B, b, C, c, then the expression **[a-d]** is equivalent to the expression [aBbCcDd].

To specify a collation item within a class, the item must be enclosed between [. and .] . Two character to one collation item mappings *must* be specified this way. For example, if the current collation rules specify that the characters "Ch" map to one character for collation purposes (as in Spanish), then this collation item would be specified as **[.Ch.]** .

To specify a group of collation items, which are classified as equal unless all other collation items in the string also match, in which case a secondary "weight" becomes significant, a single member of that group must be enclosed between [= and =] . For example, if the characters A and a are in the same group then the class expressions **[[=a=]b]**, [[=A=]b] and [Aab] are all equivalent.

The **ctype** classes can also be specified within regular expressions. These are enclosed between [: and :] . The possible **ctype** classes are:

[:alpha:]	Matches alphabetic characters
[:upper:]	Matches upper case characters
[:lower:]	Matches lower case characters
[:digit:]	Matches digits
[:alnum:]	Matches alphanumeric characters
[:space:]	Matches white space
[:print:]	Matches printable characters
[:punct:]	Matches punctuation marks
[:graph:]	Matches graphical characters
[:cntrl:]	Matches control characters

The following rules may be used to construct regular expressions from one-character regular expressions:

2.1
A one-character regular expression followed by a star (*) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

2.2
A one-character regular expression followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a regular expression that matches a *range* of occurrences of the one-character regular expression. The values of *m* and *n* must be nonnegative integers less than 255; \{*m*\} matches *exactly* *m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches any number of occurrences between *m* and *n*, inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

2.3
The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.4
A regular expression enclosed between the character sequences \(and \) is a regular expression that matches whatever the unadorned regular expression matches. See 2.6 below for a discussion of why this is useful.

2.5
The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same regular expression. Here *n* is a digit; the subexpression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression **^\\(.*\\)\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an *entire regular expression* may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A caret (^) at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.

3.2 A dollar sign ($) at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line. The construction ^*entire regular expression*$ constrains the entire regular expression to match the entire line.

The null regular expression (e.g., //) is equivalent to the last regular expression encountered.

To understand addressing in *ed* , it is necessary to know that there is a *current line* at all times. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.

2. The character **$** addresses the last line of the buffer.

3. A decimal number *n* addresses the *n*-th line of the buffer.

4. ´*x* addresses the line marked with the mark name character *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.

5. A regular expression enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.

6. A regular expression enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the regular expression. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *Files* below.

7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus or minus the indicated number of lines. The plus sign may be omitted.

8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g, **-5** is understood to mean **.-5**.

9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -.) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.

10. For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last address(es) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5 and 6 above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **p** or by **l**, in which case the current line is either printed or listed, respectively, as discussed below under the *p* and *l* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; dot is left at the address of the last inserted line, or, if there were no inserted lines, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer.

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; dot is left at the address of the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; dot is set to the last line of the buffer. If no filename is given, the currently remembered filename, if any, is used (see the *f* command). The number of characters read is typed. *file* is remembered for possible use as a default filename in subsequent *e*, *r*, and *w* commands. If *file* begins with an exclamation (!), the rest of the line is taken to be a shell command. The output of this command is read for the *e* and *r* commands. For the *w* command, the file is used as the standard input for the specified command. Such a shell command is *not* remembered as the current filename.

E *file*

The *E*dit command is like *e*, except the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *f*ilename command changes the currently remembered filename to *file*; otherwise, it prints the currently remembered filename.

(1,$)g/*regular-expression*/*command list*

In the *g*lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, the given *command list* is executed with **.** initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multiline list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted; the **.** terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *Notes* and the last paragraph before *Files* below.

(1,$)G/*regular-expression*/

In the interactive *G*lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, that line is printed, dot (**.**) is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on. A newline acts as a null command. An ampersand (&) causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by entering an INTERRUPT (pressing the DEL key).

h

The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous diagnostic if there was one. The *H* command alternately turns this mode on and off. It is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; dot is left at the address of the last inserted line, or if there were no inserted lines, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command.

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate newline characters. If only one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lowercase letter. The address ´*x* then addresses this line. Dot is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few nonprinting characters (e.g., tab, backspace) are represented by mnemonic overstrikes, all other nonprinting characters are printed in octal, and long lines are folded. An *l* command may be appended to any command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines. Dot is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character. Dot is left at the last line printed. The *n* command may be appended to any command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines. Dot is left at the last line printed. The *p* command may be appended to any command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off. It is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r ***file***

The *r*ead command reads in the given file after the addressed line. If no filename is given, the currently remembered filename, if any, is used (see *e* and *f* commands). The currently remembered filename is *not* changed unless *file* is the very first filename mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. Dot is set to the address of the last line read in. If *file* begins with !, the rest of the line is taken to be a shell command whose output is to be read. Such a shell command is *not* remembered as the current filename.

(.,.)s/*regular-expression*/*replacement*/ **or**

(.,.)s/*regular-expression*/*replacement*/g **or**

(.,.)s/*regular-expression*/*replacement*/*n* ***n*=1-512**

The *s*ubstitute command searches each addressed line for an occurrence of the specified regular expression. In each line in which a match is found, all nonoverlapped matched strings are replaced by *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or newline may be used instead of / to delimit *regular-expression* and *replacement*. Dot is left at the address of the last line on which a substitution occurred.

The *n* character represents any number between one and 512. This number indicates the instance of the pattern to be replaced on each addressed line.

An ampersand (&) appearing in *replacement* is replaced by the string matching the *regular-expression* on the current line. The special meaning of the ampersand in this context may be suppressed by preceding it with a backslash. The characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified regular expression enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or when it is preceded by a \.

A line may be split by substituting a newline character into it. The newline in the *replacement* must be escaped by preceding it with a \. Such a substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*
This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0). Dot is left at the address of the last line of the copy.

u
The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*regular-expression* **/***command list*
This command is the same as the global command *g* except that the *command list* is executed with dot initially set to every line that does *not* match the regular expression.

(1,$)V/*regular-expression* **/**
This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the regular expression.

(1,$)w *file*
The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writeable by everyone), unless the *umask* setting (see *sh*(C)) dictates otherwise. The currently remembered filename is *not* changed unless *file* is the very first filename mentioned since *ed* was invoked. If no filename is given, the currently remembered filename, if any, is used (see *e* and *f* commands), and dot remains. If the command is successful, the number of characters written is displayed. If *file* begins with an exclamation (!), the rest of the line is taken to be a shell command to which the addressed lines are supplied as the standard input. Such a shell command is *not* remembered as the current filename.

($)=
The line number of the addressed line is typed. Dot is unchanged by this command.

!*shell command*
The remainder of the line after the ! is sent to the UNIX shell (*sh*(C)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered filename. If a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed. Dot is unchanged.

(.+1)
An address alone on a line causes the addressed line to be printed. A RETURN alone on a line is equivalent to **.+1p**. This is useful for stepping forward through the editing buffer a line at a time.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a question mark (?) and returns to its command level.

ed has size limitations: 512 characters per line, 256 characters per global command list, 64 characters per filename, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a regular expression or of a replacement string (e.g., /) would be the last character before a newline, that delimiter may be omitted, in which case the addressed line is printed. Thus, the following pairs of commands are equivalent:

```
s/s1/s2     s/s1/s2/p
g/s1        g/s1/p
?s1         ?s1?
```

Files

/tmp/e# Temporary; # is the process number

ed.hup Work is saved here if the terminal is hung up

See Also

coltbl(M), grep(C), locale(M), sed(C), sh(C), stty(C), regexp(S)

Diagnostics

? Command errors
? *file* An inaccessible file

Use the *h*elp and *H*elp commands for detailed explanations.

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands by printing ? and allowing you to continue editing. A second *e* or *q* command at this point will take effect. The dash (-) command-line option inhibits this feature.

Notes

An exclamation (!) command cannot be subject to a *g* or a *v* command.

The ! command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(C)).

The sequence **\n** in a regular expression does not match any character.

The *l* command mishandles DEL.

Because 0 is an illegal address for the *w* command, it is not possible to create an empty file with *ed*.

If the editor input is coming from a command file (i.e., *ed file* < *ed-cmd-file*), the editor will exit at the first failure of a command in the command file.

Standards Conformance

ed is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

enable

turns on terminals and line printers

Syntax

enable tty ...
enable printers

Description

For terminals this program manipulates the **/etc/conf/init.base** file and signals *init* to allow logins on a particular terminal.

For line printers, *enable* activates the named printers and enables them to print requests taken by *lp*(C). Use *lpstat*(C) to find the status of the printers.

Examples

A simple command to enable **tty01** follows:

enable tty01

Files

/dev/tty*

/etc/conf/init.base

/usr/spool/lp/*

See Also

disable(C), getty(M), init(M), login(M), lp(C), lpstat(C), inittab(F), uugetty(M)

Authorization

The behavior of this utility is affected by assignment of the **printerstat** authorization, which is usually reserved for system administrators. Refer to the "Using a Trusted System" chapter of the *User's Guide* for more details.

env

sets environment for command execution

Syntax

env [-] [name=value] ... [command [args]]

Description

env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The - flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the environment is printed, one name-value pair per line.

See Also

sh(C), exec(S), profile(F), environ(M)

Notes

The old *printenv* command was replaced in and System V by the *env* command. The current *printenv* is a link to *env*.

Standards Conformance

env is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ex, edit

invokes a text editor

Syntax

ex [**-s**] [**-v**] [**-t** tag] [**-r** file] [**-L**] [**-R**] [**-c** command] name ...

edit [**-r**] [**-x**] [**-C**] name . . .

Description

ex is the root of the editors *ex* and *vi*. *ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor. It operates precisely as *ex*(C) with the following options automatically set:

novice	ON
report	ON
showmode	ON
magic	OFF

These options can be turned on or off via the *set* command in *ex*(C).

Refer to the *vi*(C) page for a complete description of the *ex* commands.

Files

/usr/lib/ex3.7strings	Error messages
/usr/lib/ex3.7recover	Recover command
/usr/lib/ex3.7preserve	Preserve command
/etc/termcap	Describes capabilities of terminals
$HOME/.exrc	Editor startup file
/tmp/Ex*nnnnn*	Editor temporary
/tmp/Rx*nnnnn*	Named buffer temporary

/usr/preserve	Preservation directory

See Also

awk(C), ctags(CP), ed(C), grep(C), sed(C), termcap(F), vi(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Standards Conformance

ex is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

expr

evaluates arguments as an expression

Syntax

expr arguments

Description

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that zero is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Expressions should be quoted, since many of the characters that have special meaning in the shell also have special meaning in *expr*. The list is in order of increasing precedence, with equal precedence operators grouped within braces ({ and }).

expr | *expr*
: Returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr & *expr*
: Returns the first *expr* if neither *expr* is null nor **0**, otherwise returns **0**.

expr { =, >, >=, <, <=, != } *expr*
: Returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison, as defined by the locale.

expr { +, - } *expr*
: Addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
: Multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*
: The matching operator : compares the first argument with the second argument which must be a regular expression; regular

expression syntax is the same as that of *ed*(C), except that all patterns are "anchored" (i.e., begin with a caret (^)) and therefore the caret is not a special character in that context. (Note that in the shell, the caret has the same meaning as the pipe symbol (|).) Normally the matching operator returns the number of characters matched (zero on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

Examples

1. a=\`expr $a + 1\`

   Adds 1 to the shell variable **a**.

2. # For $a ending in "/file"
   expr $a : '.*/\(.*\)'

   Returns the last segment of a pathname (i.e., file). Watch out for the slash alone as an argument: *expr* will take it as the division operator (see *Notes* on the next page).

3. expr $VAR : '.*'

   Returns the number of characters in **$VAR**.

See Also

coltbl(M), ed(C), locale(M), sh(C)

Diagnostics

As a side effect of expression evaluation, *expr* returns the following exit values:

0	If the expression is neither null nor zero
1	If the expression is null or zero
2	For invalid expressions

Other diagnostics include:

syntax error — For operator/operand errors, including unset variables

nonnumeric argument — If arithmetic is attempted on a nonnumeric string

Notes

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an equals sign (=), the command:

```
expr $a = =
```

looks like:

```
expr = = =
```

The arguments are passed to *expr* and will all be taken as the = operator. The following permits comparing equals signs:

```
expr X$a = X=
```

Standards Conformance

expr is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

factor

factor a number

Syntax

factor [number]

Description

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{46} (about 7.2×10^{13}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

The time it takes to factor a number, n, is proportional to $\sqrt{n}$. It usually takes longer to factor a prime or the square of a prime, than to factor other numbers.

Diagnostics

factor returns an error message if the supplied input value is greater than 2^{46} or is not an integer number.

false

returns with a nonzero exit value

Syntax

false

Description

false does nothing except return with a nonzero exit value. *true*(C), *false's* counterpart, does nothing except return with a zero exit value. "False" is typically used in shell procedures such as:

```
until false
do
        command
done
```

See Also

sh(C), true(C)

Diagnostics

false is any non-zero value.

Standards Conformance

false is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

file

determines file type

Syntax

file [**-m**] file ...

file [**-m**] **-f** namesfile

Description

file performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language.

If the **-f** option is given, *file* takes the list of filenames from *namesfile*. If the **-m** option is given, *file* sets the access time for the examined file to the current time. Otherwise, the access time remains unchanged.

Several object file formats are recognized. For **a.out** and **x.out** format object files, *file* reports "separate" if the file was linked with **cc -i**, "pure" if the file was linked with **cc -n**, and "not stripped" if the file was not linked with **cc** -s or if *strip*(CP) was not run.

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

file makes errors; in particular it often mistakes command files for C programs.

The *file* command can only distinguish English text. If an 8 bit character (a character not in the English alphabet) is found, then the text will be defined as "8 bit text".

Standards Conformance

file is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

find

finds files

Syntax

find pathname-list expression

Description

The *find* command is used to find files matching a certain set of selection criteria. *find* recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) seeking files that match a Boolean *expression* written in the primaries given below.

Expressions

For each file encountered, *find* evaluates the specified *expression*, formed of one or more of the following primary expressions, which may evaluate as true or false. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, -*n* means less than *n* and *n* means exactly *n*.

-name *file* True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for the left bracket ([), the question mark (?) and the star (*)).

-perm *onum* True if the file permission flags exactly match *onum* (see *chmod*(C)). If *onum* is prefixed by a minus sign, all other modes become significant (see *mknod*(S)), including the file type, setuid, setgid, and sticky bits rather than just read/write/execute modes for owner/group/other.

-type *x* True if the type of the file is *x*, where *x* is **b** for block special file, **c** for character special file, **d** for directory, **p** for named pipe (first-in-first-out), or **f** for regular file.

-links *n* True if the file has *n* links.

-inum *num* True if the file's inode is *num*. This is useful for locating files with matching inodes.

-user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

-group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file as a group name, it is taken as a group ID.

-size *n* True if the file is *n* blocks long (512 bytes per block).

-atime *n* True if the file was last accessed *n* days ago.

-mtime *n* True if the data in the file was last modified *n* days ago.

-ctime *n* True if the file's status was last changed (i.e. created or modified) *n* days ago.

-exec *cmd* Executes shell command *cmd*. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name. True if the executed *cmd* returns a zero value as exit status (most commands return a zero value on successful completion and a non-zero value if an error is encountered).

-ok *cmd* Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

-cpio *device* Writes the current file on *device* in *cpio*(F) format (5120-byte records). Always true.

-depth Causes all entries in a directory to be acted upon before the directory itself. This can be useful when used with *cpio*(C) or the **-cpio** expression to transfer files located in directories without write permission. Always true.

-print Causes the current path name to be printed. This option is used to create a list of all files matched by the previous primaries. Always true.

-newer *file* True if the current file has been modified more recently than the argument *file*.

(*expression*) True if the parenthesized expression is true. Usually used with the **-o** operator (see below), parentheses are used for grouping. Parentheses are special to the shell and must be escaped.

The primaries may be combined using the following operators (in order of decreasing precedence):

! The ! operator specifies the negation of the next primary (i.e., ! -newer *file* is true if the current file is *not* newer than *file*.). This is the equivalent of the logical "not" operator.

-o Placing the **-o** operator between two primaries creates an expression that is true if either of the two primaries is true. It should be used with parentheses (i.e., \(-perm 644 -o -perm 664 \) is true if the current file has permissions 644 or 664). This is equivalent to the logical "inclusive or" operator.

Note that placing two primaries next to each other is the equivalent of the logical "and" operation. The precedence of this operation is less than that of the ! operator but greater than that of the **-o** operator.

Examples

The following command searches for files named *chapter1* in the current directory and all directories below it and sends the pathname of any such files it finds to the standard output:

```
find . -name chapter1 -print
```

The following removes all files named **core** or with names ending in **.out** that have not been accessed in the last seven days.

```
find / \( -name core -o -name "*.out" \) -atime +7 -exec rm {} \;
```

Files

/etc/passwd	User names and uids
/etc/group	Group names and gids

See Also

cpio(C)(F), sh(C), stat(S), test(C)

Standards Conformance

find is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

finger

finds information about users

Syntax

finger [**-bfilpqsw**] [login1 [login2 ...]]

Description

By default *finger* lists the login name, full name, terminal name and write status (as a "*" before the terminal name if write permission is denied), idle time, login time, office location, and phone number (if they are known) for each current user. (Idle time is minutes if it is a single integer, hours and minutes if a colon (:) is present, or days and hours if a "d" is present.)

A longer format also exists and is used by *finger* whenever a list of names is given. (Account names as well as first and last names of users are accepted.) This is a multiline format; it includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* which is also in the home directory.

finger options are:

-b Briefer long output format of users.

-f Suppresses the printing of the header line (short format).

-i Quick list of users with idle times.

-l Forces long output format.

-p Suppresses printing of the *.plan* files.

-q Quick list of users.

-s Forces short output format.

-w Forces narrow format list of specified users.

Files

/etc/utmp	Who file
/etc/passwd	User names, offices, phones, login directories, and shells
$HOME/.plan	Plans
$HOME/.project	Projects

See Also

who(C), w(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Only the first line of the *.project* file is printed.

Entries in the /etc/passwd file have the following format:

login name:user password(coded):user ID:group ID:comments:home directory:login shell

The comment field corresponds to what appears in the *finger* output. For example, in the following **/etc/passwd** entry:

```
blf:*:47:5:Brian Foster, Mission, x70, 767-1234
:/u/blf:/bin/sh
```

the comment field, "Brian Foster, Mission, x70, 767-1234" , contains data for the "In Real Life" , "Office" , and "Home Phone" , columns of the *finger* listings.

Idle time is computed as the elapsed time since any activity on the given terminal. This includes previous invocations of *finger* which may have modified the terminal's corresponding device file **/dev/tty??**.

fixhdr

changes executable binary file headers

Syntax

fixhdr option files

Description

fixhdr changes the header of output files created by link editors or assemblers. The kinds of modifications include changing the format of the header, the fixed stack size, the standalone load address, and symbol names.

Using *fixhdr* allows the use of binary executable files, created under other versions or machines, by simply changing the header information so that it is usable by the target cpu.

These are the options to *fixhdr* :

-xa Change the *x.out* format of the header to the *a.out* format.

-xb Change the *x.out* format of the header to the *b.out* format.

-x4 Change the *x.out* format of the header to the 4.2BSD *a.out* format.

-x5 [**-n**] Change the *x.out* format of the header to Altos UNIX System V *a.out* format. The **-n** flag causes leading underscores on symbol names to be passed with no modifications.

-ax -c [11,86]
Change the *a.out* format of the header to the *x.out* format. The **-c** flag specifies the target CPU. 11 specifies a PDP-11 CPU. 86 specifies one of the 8086 family of CPUs (8086, 8088, 80186, 80286 or 80386).

-bx Change the *b.out* format of the header to the *x.out* format.

-5x [**-n**] Change the Altos UNIX System V *a.out* format of the header to the *x.out* format. The **-n** flag causes leading underscores on symbol names to be passed with no modifications.

-86x Add the *x.out* header format to the *86rel* object module format. See *86rel*(F).

-F num Add (or change) the fixed stack size specified in the *x.out* format of the header. *num* must be a hexadecimal number.

-A num Add (or change) the standalone load address specified in the *x.out* format of the header. *num* must be a hexadecimal number.

-M[smlh] Change the model of the *x.out* or *86rel* format. Model refers to the compiler model specified when creating the binary. **s** refers to small model, **m** refers to medium model, **l** refers to large model, and **h** refers to huge model.

-v [2,3,5,7] Change the version of XENIX specified in the header. XENIX version 2 was based on UNIX Version 7.

-s s1=s2 [-s s3=s4]
Change symbol names, where symbol name *s1* is changed to *s2*.

-r Ensure that the resolution table is of non-zero size.

-C cpu Set the *cpu* type. *cpu* can be 186, 286, 386, 8086, others.

Files

/usr/bin/fixhdr

See Also

a.out(F), 86rel(F)

Notes

Give *fixhdr* one option at a time. If you need to make more than one kind of modification to a file, use *fixhdr* on the original file. Then use it again on the *fixhdr* output, specifying the next option. Copy the original file if you need an unmodified version as *fixhdr* makes the modifications directly to the file.

Value Added

fixhdr is an extension of AT&T System V provided by Altos UNIX System V.

format

format floppy disks and mini-cartridge tapes

Syntax

format [**-n**] [**-v**] [**-f**] [**-q**] [device] [**-i** interleave]

Description

format formats diskettes for use on an Altos UNIX System V system. It may be used either interactively or from the command line. The default drive is **/dev/rfd0**.

Options

The following command line options are available:

-f Suppresses the interactive feature. The *format* program does not wait for user-confirmation before starting to format the diskette. Regardless of whether or not you run *format* interactively, track and head information is displayed.

device
This specifies the device to be formatted. The default device is **/dev/rfd0** .

-i *interleave*
Specifies the interleave factor.

-q Quiet option. Suppresses the track and head output information normally displayed. Although this option does not suppress the interactive prompt, it would typically be used with **-f** to produce no output at all.

-v Specifies format verification.

-n Specifies that the diskette is not to be verified (overrides verify entry in **/etc/default/format**).

The file **/etc/default/format** is used to specify the default device to be formatted and whether or not each diskette is to be verified. The entries must be in the format DEVICE=/dev/rfd*nnn* and VERIFY=[yYnN], as in the following example:

```
DEVICE=/dev/rfd096ds15
VERIFY=y
```

The device must be a character (raw) device.

Usage

To run *format* interactively, enter:

format

followed by any of the legal options except **-f**, and press RETURN. When you run *format* interactively, you see the prompt:

```
insert diskette in drive and press return when ready
```

When you press RETURN at this prompt, *format* begins to format the diskette.

If you specify the **-f** option, you do not see this prompt. Instead, the program begins formatting immediately upon invocation.

Unless you specify the **-q** option, *format* displays which track and head it is currently on:

```
track #   head #
```

The number signs above are replaced by the actual track and head information.

Files

/etc/default/format

/dev/rfd[0 - n]

See Also

fd(HW)

Notes

The *format* utility does not format floppies for use under DOS; use the *dosformat* command documented in *dos*(C).

Altos UNIX System V systems require error free floppies.

It is not advisable to format a low density (48tpi) diskette on a high density (96tpi) floppy drive. Diskettes written on a high density drive should be read on high density drives. A low density diskette written on a high density drive may not be readable on a low density drive.

The device */dev/install* is used only for installing and reading floppies. Attempts made to format this device may result in an error.

fuser

Identify processes using a file or file structure

Syntax

/etc/fuser [**-ku**] *file*... | *resource*... [-] [[**-ku**] *file*... | *resource*...]

Description

The *fuser* utility displays the process IDs of processes that are using the *file*s or remote *resource*s specified as arguments. Each process ID is followed by a letter code, interpreted as follows if the process is using the file as:

c current directory

p parent of its current directory (only when the file is being used by the system)

r root directory

For block special devices with mounted filesystems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource (Remote File Sharing) are reported. (*fuser* cannot use the mount point of the remote resource; it must use the resource name.) For all other types of files (e.g., text files, executables, directories, devices), only the processes using the specified file are reported.

The following options may be used with *fuser*:

-u The user login name, in parentheses, also follows the process ID.

-k The SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately (see *kill*(S)).

If more than one group of files is specified, the options any be specified again for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single newline. All other output is written to standard error.

You cannot list processes using a particular file from a remote resource mounted on your machine. You can only use the resource name as an argument.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use *fuser*. Only the superuser can terminate another user's process, however.

Files

/unix	for system namelist
/dev/kmem	for system image
/dev/mem	also for system image

See Also

mount(ADM), ps(C), kill(S), signal(S)

getopt

parses command options

Syntax

set -- `getopt** optstring **$*`

Description

getopt is used to check and break up options in command lines for parsing by shell procedures. *optstring* is a string of recognized option letters (see *getopt* (S)). If a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by whitespace. The special option **--** is used to delimit the end of the options. *getopt* will place **--** in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments ($1 $2 . . .) are reset so that each option is preceded by a dash (-) and in its own shell argument. Each option argument is also in its own shell argument.

Example

The following code fragment shows how one can process the arguments for a command that can take the options **a** and **b**, and the option **o**, which requires an argument:

```
set - - `getopt abo: $*`
if [ $? != 0 ]
then
        echo "usage: $0 [-a | -b] [-o <arg>]"
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b) shift; FLAG=$i;;
        -o)          OARG=$3; shift; shift;;
        - -)         shift;  break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg
cmd -a -o arg
cmd -oarg -a
cmd -a -oarg --
```

See Also

sh(C), getopt(S)

Diagnostics

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

Notes

The "Syntax" given for this utility assumes the user has a *sh*(C) shell.

Standards Conformance

getopt is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

getopts, getoptcvt

parses command options

Syntax

getopts optstring name [arg . . .]

/usr/lib/getoptcvt [**-b**] file

Description

The *getopts* command is used by shell procedures to parse positional parameters and to check for legal options. It supports all applicable rules of the command syntax standard [see Rules 3-10, *intro* (C)]. It should be used in place of the *getopt*(C) command. (See the **Notes** below.)

This feature is only available in the Bourne shell.

optstring must contain the option letters the command using *getopts* will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, *getopts* will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable OPTIND. Whenever the shell or a shell procedure is invoked, OPTIND is initialized to 1.

When an option requires an option-argument, *getopts* places it in the shell variable **OPTARG.**

If an illegal option is encountered, ? will be placed in *name*.

When the end of options is encountered, *getopts* exits with a non-zero exit status. The special option "--" may be used to delimit the end of the options.

By default, *getopts* parses the positional parameters. If extra arguments (*arg* . . .) are given on the *getopts* command line, *getopts* will parse them instead.

The */usr/lib/getoptcvt* command reads the shell script in *file*, converts it to use *getopts* (C) instead of *getopt* (C), and writes the results to the standard output.

-b the results of running */usr/lib/getoptcvt* will be portable to earlier UNIX releases. */usr/lib/getoptcvt* modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke *getopts*(C) or *getopt*(C).

So all new commands will adhere to the command syntax standard described in *intro*(C), they should use *getopts*(C) or *getopt*(S) to parse positional parameters and check for options that are legal for that command (see **Notes** below).

Examples

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
     case $c in
     a | b)     FLAG=$c;;
     o)         OARG=$OPTARG;;
     ?)         echo $USAGE
                exit 2;;
     esac
done
shift `expr $OPTIND - 1`
```

This code will accept any of the following as equivalent:

```
cmd -a -b -o "xxx z yy"
cmd -a -b -o "xxx z yy" --
cmd -ab -o xxx,z,yy
cmd -ab -o "xxx z yy"
cmd -o xxx,z,yy -b -a
```

See Also

intro(C), sh(C), getopt(S)

Notes

Although the following command syntax rule [see *Intro*(C)] relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **Examples** section above, **a** and **b** are options, and the option **o** requires an option-argument:

cmd -aboxxx file (Rule 5 violation: options with option-arguments must not be grouped with other options.)

cmd -ab -oxxx file (Rule 6 violation: there must be white space after an option that takes an option-argument.)

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

Diagnostics

getopts prints an error message to the standard error when it encounters an option letter not included in *optstring*.

gets

gets a string from the standard input

Syntax

gets [string]

Description

gets can be used with *csh*(C) to read a string from the standard input. If *string* is given it is used as a default value if an error occurs. The resulting string (either *string* or as read from the standard input) is written to the standard output. If no *string* is given and an error occurs, *gets* exits with exit status 1.

See Also

line(C), csh(C)

greek

select terminal filter

Syntax

greek [-Tterminal]

Description

greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE Model 37 terminal for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable **$TERM** [see *environ*(M)]. Currently, the following *terminal*s are recognized:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

Files

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

See Also

300(C), 4014(C), 450(C), hp(C), tplot(ADM), environ(M), term(M)

grep, egrep, fgrep

searches a file for a pattern

Syntax

grep [**-bchlnsvy**] [**-e** expression] [files]

egrep [**-bchlnv**] [**-e** expression] [files]

fgrep [**-bclnvxy**] [**-f** expfile] [files]

Description

Commands of the *grep* family search the input *files* (or standard input if no *files* are specified) for lines matching a pattern. Normally, each matching line is copied to the standard output. If more than one file is being searched, the name of the file in which each match occurs is also written to the standard output along with the matching line (unless the **-h** option is used, see below).

grep patterns are limited regular *expressions* in the style of *ed*(C). *grep* uses a compact nondeterministic algorithm. *egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *fgrep* patterns are fixed *strings*. *fgrep* is fast and compact. The following *options* are recognized:

-v All lines but those matching are displayed.

-x Displays only exact matches of an entire line. (*fgrep* only.)

-c Only a count of matching lines is displayed.

-l Only the names of files with matching lines are displayed, separated by newlines.

-h Prevents the name of the file containing the matching line from being prepended to that line. Used when searching multiple files. (This option works with *grep* and *egrep* only.)

-n Each line is preceded by its relative line number in the file.

-b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

-s Suppresses error messages produced for nonexistent or unreadable files. (*grep* only). Note that the **-s** option will not suppress error messages generated by the **-f** option.

-y Turns on matching of letters of either case in the input so that case is insignificant. Conversion between uppercase and lowercase letters is dependent on the locale setting. **-y** does not work with *egrep*.

-e expression or*strings*
Same as a simple *expression* argument, but useful when the *expression* begins with a dash (-).

-f *expfile*
The regular *expression* for *grep* or *egrep*, or *strings* list for *fgrep* is taken from the *expfile*.

In all cases (except with -h) the filename is output if there is more than one input file. Care should be taken when using the characters $, *, [, ^, |, (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* or *strings* argument in single quotation marks. For example:

```
grep '[Ss]omeone' text.file
```

This command would find all lines containing the word "someone" in the file **text.file**, whether the initial "s" is uppercase or lowercase.

Multiple strings can be specified in *fgrep* without using a separate strings file by using the quoting conventions of the shell to imbed newlines in the *string* argument. For example, if you were using the Bourne shell (*sh*(C)) you might enter the following on the command line:

```
fgrep 'Someone
someone' text.file
```

This would have the same effect as the *grep* example above. See the *csh*(C) manual page for ways to imbed newlines in a string when using *csh*(C).

egrep accepts regular expressions as in *ed*(C), with the addition of the following:

- A regular expression followed by a plus sign (+) matches one or more occurrences of the regular expression.

- A regular expression followed by a question mark (?) matches 0 or 1 occurrences of the regular expression.

- Two regular expressions separated by a vertical bar (|) or by a newline match strings that are matched by either regular expression.

- A regular expression may be enclosed in parentheses ( ) for grouping. For example:

  egrep ´([Ss]ome|[Aa]ny)one´ text.file

This example displays all lines in **text.file** containing the words "someone" or "anyone", whether or not they are spelled with initial capital letters. Without the parentheses, this example would display all lines containing the words "some" or "anyone" (because the vertical bar (|) operator is of lower precedence than concatenation, see below).

Because of the algorithm used, *egrep* does not support extended ranges as in *ed*(C): Ranges like [a-z] are interpreted on the basis of the machine's collating sequence, not the collating sequence defined by the locale. *grep* supports *col*(C) extended ranges.

The \(and \) operators, supported by *ed*(C), are not supported by *egrep*.

The order of precedence of operators is [ ], then * ? +, then concatenation, then backslash (\) with newline or vertical bar (|).

See Also

coltbl(M), ed(C), locale(M), sed(C), sh(C)

Diagnostics

Exit status is 0 if any matches are found, 1 if no matches are found, and 2 for syntax errors or inaccessible files.

Notes

Ideally there should be only one *grep*, but there isn't a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters. Longer lines are truncated.

When using *grep* with the -y option, the search is not made totally case insensitive in character ranges specified within brackets.

Standards Conformance

egrep, *fgrep* and *grep* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

grpcheck

checks group file

Syntax

grpcheck [file]

Description

grpcheck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

Files

/etc/group

/etc/passwd

See Also

pwcheck(C), group(F), passwd(F)

Diagnostics

Group entries in **/etc/group** with no login names are flagged.

hd

displays files in hexadecimal format

Syntax

hd [-format [**-s** offset] [**-n** count] [file] ...

Description

The *hd* command displays the contents of files in hexadecimal, octal, decimal, and character formats. Control over the specification of ranges of characters is also available. The default behavior is with the following flags set: "-abx -A". This says that addresses (file offsets) and bytes are printed in hexadecimal and that characters are also printed. If no *file* argument is given, the standard input is read.

Options include:

-s *offset* Specify the beginning offset in the file where printing is to begin. If no 'file' argument is given, or if a seek fails because the input is a pipe, 'offset' bytes are read from the input and discarded. Otherwise, a seek error will terminate processing of the current file.

The *offset* may be given in decimal, hexadecimal (preceded by '0x'), or octal (preceded by a '0'). It is optionally followed by one of the following multipliers: **w**, **l**, **b**, or **k**; for words (2 bytes), long words (4 bytes), half kilobytes (512 bytes), or kilobytes (1024 bytes), respectively. Note that this is the one case where "b" does *not* stand for bytes. Since specifying a hexadecimal offset in blocks would result in an ambiguous trailing 'b', any offset and multiplier may be separated by an asterisk (*).

-n *count* Specify the number of bytes to process. The *count* is in the same format as *offset*, above.

Format Flags

Format flags may specify addresses, characters, bytes, words (2 bytes) or longs (4 bytes) to be printed in hex, decimal, or octal. Two special formats may also be indicated: text or ascii. Format and base specifiers may be freely combined and repeated as desired in order to specify different bases (hexadecimal, decimal or octal) for different output formats (addresses, characters, etc.). All format flags appearing in a single argument are applied as appropriate to all other flags in that argument.

acbwlA

Output format specifiers for addresses, characters, bytes, words, longs and ascii respectively. Only one base specifier will be used for addresses. The address will appear on the first line of output that begins each new offset in the input.

The character format prints all printable characters without change, special C escapes as defined in the language, and the remaining values in the specified base.

The ascii format prints all printable characters without change, and all others as a period (.). This format appears to the right of the first of other specified output formats. A base specifier has no meaning with the ascii format. If no other output format (other than addresses) is given, **bx** is assumed. If no base specifier is given, *all* of **xdo** are used.

xdo

Output base specifiers for hexadecimal, decimal and octal.

t Print a text file, each line preceded by the address in the file. Normally, lines should be terminated by a **\n character;** but long lines will be broken up. Control characters in the range 0x00 to 0x1f are printed as '^@' to '^_'. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special characters (^, ~, \) are preceded by a backslash (\) to escape their special meaning. As special cases, these two values are represented numerically as '\177' and '\377'. This flag will override all output format specifiers except addresses.

If no output format is given, but a base specifier is present, the output format is set to **-acbwl**. If no base specifier is given, but an output format is present, the base specifier is set to **-xdo**. If neither is present, the format flag is set to **-abx -A**.

Value Added

hd is an extension of AT&T System V provided in Altos UNIX System V.

head

prints the first few lines of a file

Syntax

head [-count] [file ...]

Description

This filter prints the first *count* lines of each of the specified files. If no files are specified, *head* reads from the standard input. If no *count* is specified, then 10 lines are printed.

See Also

tail(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

hello

send a message to another user

Syntax

hello user [tty]

Description

hello sends messages from one user to another. When first called, *hello* displays the following message:

Message from *sender's-system! sender's-name sender's-tty*

The recipient of the message should write back at this point. Communication continues until an interrupt is sent. (On most terminals, pressing the **Del** key sends an interrupt.) At that point *hello* prints "(end of message)" on the other terminal, and exits.

To write to a user who is logged in more than once, the user can employ the *tty* argument to specify the appropriate terminal name. The *who*(C) command can be used to determine the correct terminal name.

Permission to write may be allowed or denied by the recipient, using the *mesg* command. Writing is allowed by default. Certain commands, such as *nroff* and *pr*, prohibit messages in order to prevent disruption of output.

If the character **!** is found at the beginning of a line, *hello* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *hello*. When first writing to another user, the sender should wait for that user to write back before sending a message. Each party should end each message with a signal indicating that the other may reply: **o** for "over" is conventional. The signal **oo** for "over and out" is suggested when conversation is about to be terminated.

Files

/etc/utmp
/bin/sh

See Also

mesg(C), who(C), mail(C)

Value Added

hello is an extension of AT&T System V provided in Altos UNIX System V.

hostid

Print unique hardware ID

Syntax

hostid

Description

The *hostid* utility prints the system's unique hardware ID to standard output. This ID is set at the factory during manufacture.

If a hardware ID is not available (for example, on a non-Altos machine), the operating system serial number is returned instead.

Value Added

hostid is an extension of AT&T System V provided by Altos UNIX System V.

hp

handle special functions of Hewlett-Packard terminals

Syntax

hp [-e] [-m]

Description

hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff* output. A typical usage is in conjunction with text processing software:

nroff -h files ... | hp

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags are as follows:

-e It is assumed that your terminal has the "display enhancements" feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underlined mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark-on-light, rather than the usual light-on-dark.

-m
Requests minimization of output by changing newlines to ^M's. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set as *300*(C), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown.

Diagnostics

line too long if the representation of a line exceeds 1,024 characters. The exit codes are **0** for normal termination, **2** for all errors.

See Also

300(C), greek(C)

Notes

An ''overstriking sequence'' is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text ''disappear''. In particular, tables generated by *tbl*(CT) that contain vertical lines will often be missing the lines of text that contain the ''foot'' of a vertical line, unless the input to *hp* is piped through *col*(C).

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

hwconfig

read the configuration information

Syntax

/etc/hwconfig [-nlh] [**-f** filename] [[-] param] [[-] param=val] ...

Description

hwconfig returns the configuration information contained in the file **/usr/adm/hwconfig** or the file specified on the command line with the **-f** *filename* option. Using combinations of the remaining options, the user can view as much information as needed from the configuration file. The display format is as follows:

magic_char device_name base+finish vec dma rest

where:

magic_char	is the character %
device_name	is the name of the device driver
base+finish	is the starting and the finishing addresses of the driver working space
vec	is the interrupt vector number
dma	is the dma channel number
rest	is a possibly empty list of parameter=value pairs

The default *hwconfig* display looks similar to this:

```
device    address        vector dma comment
--------------------------------------------------------------------
fpu       -              35     -   type=80387
floppy    0x3F2-0x3F7    06     2   unit=0 type=96ds15
serial    0x2F8-0x2FF    03     -   unit=1 type=Standard nports=1
parallel  0x378-0x37A    07     -   unit=0
console   -              -      -   unit=ega type=0
disk      0x1F0-0x1F7    36     -   type=W0 unit=0 cyls=791 hds=16 secs=48
```

Options

-n	the device name is always printed out.

-l the long format of the device configuration content is used.

-h uses the long format, with headers.

-f*file* uses *file* as the input file instead of the default **/usr/adm/hwconfig.**

param any of the 12 pre-defined parameters available: name, base, offset, vec, dma, unit, type, nports, hds, cyls, secs, and drvr.

-param shows all values of *param* throughout the configuration file. *param* can be any valid system parameter.

-param=val shows only information from the line where *param* equals the value *val.*

The **-n**, **-l** and **-h** options are in increasing overriding power. That is, if **-n** and **-l** are both specified, **-l** will be used. *param* on its own indicates a query for its corresponding value(s), whereas *param=value* indicates a matching <token,val> pair in the input file. **-l** is used by default if there are no queries and no explicit option.

Command line queries, i.e. those with parameters only, are always displayed in short format.

Examples

hwconfig The entire contents of the file **/usr/adm/hwconfig** is printed.

hwconfig base
All the values of the *base* parameter found in **/usr/adm/hwconfig** are printed.

hwconfig -f conf base=300 vec=19
All entries in *conf* that match the *base* and *vec* values given are printed.

hwconfig name=floppy base
The name and value of *base* in **/usr/adm/hwconfig** for the drivers with the name *floppy* are printed for all entries.

hwconfig -n base dma
The device name associated with the *base* and *dma* is displayed. For example:

```
name=scsi base=0x234 dma=4
```

hwconfig base dma vec=4
The base and dma values of all **/usr/adm/hwconfig** entries with matching vec=4 are printed.

hwconfig -l base dma vec=4
is like

hwconfig -l vec=4
except that base and dma values will be printed first.

hwconfig -h
Everything is printed in the long format, with a header similar to the one shown at boot-up time. It will ignore all queries, but do matching on the token values. For example,

```
hwconfig -h vec=4 dma=1
```

will print in long format, with header, all those entries with vec=4 and dma=1

Files

/usr/adm/hwconfig

Value Added

hwconfig is an extension of AT&T System V provided in Altos UNIX System V.

i286emul

emulate UNIX 80286

Syntax

i286emul [arg ...] prog286

Description

i286emul is an emulator that allows programs from UNIX System V Release 2 or Release 3 on the Intel 80286 to run on UNIX System V Release 3 on the Intel 80386.

The Altos UNIX System V system recognizes an attempt to *exec*(S) a 286 program, and automatically *exec's* the 286 emulator with the 286 program name as an additional argument. It is not necessary to specify the *i286emul* emulator on the command line. The 286 programs can be invoked using the same command format as on the 286 UNIX System V.

i286emul reads the 286 program's text and data into memory and maps them through the LDT (Local Descriptor Table) (via *sysi86*(S)) as 286 text and data segments. It also sets callgate 89 in the GDT (Global Descriptor Table) (which is used by 286 programs for system calls) to point to a routine in *i286emul*. *i286emul* starts the 286 program by jumping to its entry point.

When the 286 program attempts to do a system call, *i286emul* takes control. It does any conversions needed between the 286 system call and the equivalent 386 system call, and performs the 386 system call. The results are converted to the form the 286 program expects, and the 286 program is resumed.

The following are some of the differences between a program running on a 286 and a 286 program using *i286emul* on a 386:

- A 286 program under *i286emul* always has 64K in the stack segment if it is a large-model process, or 64K in the data segment if it is a small-model process.

- System calls and signal handling use more space on the stack under *i286emul* than it does on a 286.

- Attempts to unlink or write on the 286 program will fail on the 286 with ETXTBSY. Under *i286emul*, they will not fail.

- *ptrace*(S) is not supported under *i286emul*.

- The 286 program must be readable for the emulator to read it.

Files

/bin/i286emul
The emulator must have this name and be in **/bin** if it is to be automatically invoked when *exec* (S) is used on a 286 program.

Notes

The signal mechanism under the emulator is the System V release 2 signal mechanism rather than the System V release 3 mechanism.

id

prints user and group IDs and names

Syntax

id

Description

id writes a message on the standard output, giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

See Also

logname(C), getuid(S)

Standards Conformance

id is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ismpx

return windowing terminal state

Syntax

ismpx [-s]

Description

The *ismpx* command reports whether its standard input is connected to a multiplexed *xt*(HW) channel; i.e., whether it's running under *layers*(C) or not. It is useful for shell scripts that download programs to a windowing terminal or depend on screen size.

The *ismpx* command prints **yes** and returns **0** if invoked under *layers*(C), and prints **no** and returns **1** otherwise.

-s Do not print anything; just return the proper exit status.

Diagnostics

Returns **0** if invoked under *layers*(C), **1** if not.

See Also

jwin(C), layers(C), xt(HW)

Example

```
if ismpx -s
then
        jwin
fi
```

join

joins two relations

Syntax

join [options] file1 file2

Description

join prints to the standard output a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is a dash (-), the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

-a*n* In addition to the normal output, produces a line for each unpairable line in file *n*, where *n* is 1 or 2.

-e *s* Replaces empty output fields by string *s*.

-j*n m* Joins on the *m*th field of file *n*. If *n* is missing, uses the *m*th field in each file.

-o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number.

-t*c* Uses character *c* as a field separator. Every appearance of *c* in a line is significant.

See Also

awk(C), comm(C), sort(C)

Notes

With default field separation, the collating sequence is that of **sort -b**. With **-t**, the sequence is that of a plain sort.

Standards Conformance

join is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

jterm

reset layer of windowing terminal

Syntax

jterm

Description

The *jterm* command is used to reset a layer of a windowing terminal after downloading a terminal program that changes the terminal attributes of the layer. It is useful only under *layers*(C). In practice, it is most commonly used to restart the default terminal emulator after using an alternate one provided with a terminal-specific application package. For example, on the AT&T TELETYPE 5620 DMD terminal, after executing the *hp2621*(C) command in a layer, issuing the *jterm* command will restart the default terminal emulator in that layer.

Diagnostics

Returns **0** upon successful completion, **1** otherwise.

Notes

The layer that is reset is the one attached to standard error; that is, the window you are in when you type the *jterm* command.

See Also

layers(C)

jwin

print size of layer

Syntax

jwin

Description

The *jwin* command runs only under *layers*(C) and is used to determine the size of the layer associated with the current process. It prints the width and the height of the layer in bytes (number of characters across and number of lines, respectively). For bit-mapped terminals only, it also prints the width and height of the layer in bits.

Diagnostics

Returns **0** on successful completion, **1** otherwise.

If *layers*(C) has not been invoked, an error message is printed:

```
jwin: not mpx
```

Note

The layer whose size is printed is the one attached to standard input; that is, the window you are in when you type the *jwin* command.

See Also

layers(C)

Example

In the following example, the user input is in bold:

```
$jwin
bytes:  86 25
bits:   780 406
```

kill

terminates a process

Syntax

kill [-signo] processid ...

Description

kill sends signal 15 (terminate) to the specified process(es). This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process (background process) started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(C).

For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by - is given as the first argument, that signal is sent instead of the terminate signal (see *signal*(S)). In particular "kill -9 ..." is a sure kill.

See Also

ps(C), sh(C), kill(S), signal(S)

Standards Conformance

kill is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ksh, rksh

KornShell, a standard/restricted command and programming language

Syntax

ksh [**±aefhikmnoprstuvx**] [±**o** option] ... [**-c** string] [arg ...]
rksh [**±aefhikmnoprstuvx**] [±**o** option] ... [**-c** string] [arg ...]

Description

ksh is a command and programming language that executes commands read from a terminal or a file. *rksh* is a restricted version of the command interpreter *ksh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Definitions

A *metacharacter* is one of the following characters:

; & ( ) | < > new-line space tab

A *blank* is a **tab** or a **space**. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *functions* and *named parameters*. A *word* is a sequence of *characters* separated by one or more non-quoted *metacharacters*.

A *command* is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A special command is a command that is carried out by the shell without creating a separate process. Except for documented side effects, most special commands can be implemented as separate utilities.

Commands

A *simple-command* is a sequence of *blank* separated words which may be preceded by a parameter assignment list. (See *Environment* below). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(S)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(S) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a *pipe*(S) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;**, **&**, or **|&**. Of these five symbols, **;**, **&**, and **|&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **|&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent Shell using the **-p** option of the special commands **read** and **print** described later. The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines may appear in a *list,* instead of a semicolon, to delimit a command.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *identifier* [**in** *word* ...] **;do** *list* **;done**
Each time a **for** command is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

select *identifier* [**in** *word* ...] **;do** *list* **;done**
A **select** command prints on standard error (file descriptor 2), the set of *words*, each preceded by a number. If **in** *word* ... is omitted, then the positional parameters are used instead (see *Parameter Substitution* below). The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed *words*, then the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the parameter *identifier* is set to *null*. The contents of the line read from standard input is saved in the parameter **REPLY**. The *list* is executed for each selection until a **break** or *end-of-file* is encountered.

case *word* **in** [[( ]*pattern* [| *pattern*] ...) *list* **;;**] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that

used for file-name generation (see *File Name Generation* below).

if *list* **;then** *list* [**elif** *list* **;then** *list*] ... [**;else** *list*] **;fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **;do** *list* **;done**
until *list* **;do** *list* **;done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)
Execute *list* in a separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{ *list*;}
list is simply executed. Note that unlike the metacharacters (and), { and } are *reserved words* and must at the beginning of a line or after a ; in order to be recognized.

[[*expression*]]
Evaluates *expression* and returns a zero exit status when *expression* is true. See *Conditional Expressions* below, for a description of *expression.*

function *identifier* { *list* ;}
identifier () { *list* ;}
Define a function which is referenced by *identifier*. The body of the function is the *list* of commands between { and }. (See *Functions* below).

time *pipeline*
The *pipeline* is executed and the elapsed time as well as the user and system time are printed on standard error.

The following reserved words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { } function select time [[ ]]

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Aliasing

The first word of each command is replaced by the text of an **alias** if an **alias** for this word has been defined. The first character of an **alias** name can be any non-special printable character, but the rest of the characters must be the same as for a valid *identifier*. The replacement string can contain any valid Shell script including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for aliases. If the last character of the alias value is a *blank* then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the reserved words listed above. Aliases can be created, listed, and exported with the **alias** command and can be removed with the **unalias** command. Exported aliases remain in effect for scripts invoked by name, but must be reinitialized for separate invocations of the Shell (See *Invocation* below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the **alias** definition command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the corresponding command is looked up and becomes undefined each time the **PATH** variable is reset. These aliases remain *tracked* so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The **-h** option of the set command makes each referenced command name into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

autoload=´typeset -fu´
false=´let 0´
functions=´typeset -f´
hash=´alias -t´
history=´fc -l´
integer=´typeset -i´
nohup=´nohup ´
r=´fc -e -´

true=':'
type='whence -v'

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, then the word up to a / is checked to see if it matches a user name in the **/etc/passwd** file. If a match is found, the ~ and the matched login name is replaced by the login directory of the matched user. This is called a *tilde* substitution. If no match is found, the original text is left unchanged. A ~ by itself, or in front of a /, is replaced by the value of the **HOME** parameter. A ~ followed by a + or - is replaced by **$PWD** and **$OLDPWD** respectively.

In addition, *tilde* substitution is attempted when the value of a *variable assignment parameter* begins with a ~.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign ($()) or a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See *Quoting* below). The command substitution **$(cat file)** can be replaced by the equivalent but faster **$(<file)**. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

An arithmetic expression enclosed in double parenthesis preceded by a dollar sign ($(())) is replaced by the value of the arithmetic expression within the double parenthesis.

Parameter Substitution

A *parameter* is an *identifier*, one or more digits, or any of the characters *, @, #, ?, -, $, and !. A *named parameter* (a parameter denoted by an identifier) has a *value* and zero or more *attributes*. *Named parameters* can be assigned *values* and *attributes* by using the **typeset** special command. The attributes supported by the Shell are described later with the **typeset** special command. Exported parameters pass values and attributes to the environment.

The shell supports a one-dimensional array facility. An element of an array parameter is referenced by a *subscript*. A *subscript* is denoted by a [, followed by an *arithmetic expression* (see Arithmetic evaluation below) followed by a]. To assign values to an array, use **set -A** *name value* .... The value of all subscripts must be in the range of 0 through 1023. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the element zero.

The *value* of a *named parameter* may also be assigned by writing:

name=*value* [*name*=*value*] ...

If the integer attribute, **-i**, is set for *name* the *value* is subject to arithmetic evaluation as described below.
Positional parameters, parameters denoted by a number, may be assigned values with the **set** special command. Parameter **$0** is set from argument zero when the shell is invoked.
The character **$** is used to introduce substitutable *parameters*.

${*parameter***}**
The shell reads all the characters from **${** to the matching **}** as part of the same word even if it contains braces or metacharacters. The value, if any, of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is ***** or **@**, then all the positional parameters, starting with **$1**, are substituted (separated by a field separator character). If an array *identifier* with subscript * or @ is used, then the value for each of the elements is substituted (separated by a field separator character).

${#*parameter***}**
If *parameter* is * or @, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

${#*identifier***[*]}**
The number of elements in the array *identifier* is substituted.

${*parameter***:-***word***}**
If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

${*parameter***:=***word***}**
If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**
If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter***:+***word***}**
If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

${*parameter***#***pattern***}**
${*parameter***##***pattern***}**
If the Shell *pattern* matches the beginning of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise the value of this

parameter is substituted. In the first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted.

${*parameter* **%***pattern***}**
${*parameter* **%%***pattern***}**
If the Shell *pattern* matches the end of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched part deleted; otherwise substitute the value of *parameter*. In the first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-$(pwd)}
```

If the colon (**:**) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last executed command.

$ The process number of this shell.

_ Initially, the value _ is an absolute pathname of the shell or script being executed as passed in the *environment*. Subsequently it is assigned the last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail.

! The process number of the last background command invoked.

ERRNO
The value of *errno* as set by the most recently failed system call. This value is system dependent and is intended for debugging purposes.

LINENO
The line number of the current line within the script or function being executed.

OLDPWD
The previous working directory set by the **cd** command.

OPTARG
The value of the last option argument processed by the **getopts** special command.

OPTIND
The index of the last option argument processed by the **getopts** special command.

PPID
The process number of the parent of the shell.

PWD
The present working directory set by the **cd** command.

RANDOM
Each time this parameter is referenced, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.

REPLY
This parameter is set by the **select** statement and by the **read** special command when no arguments are supplied.

SECONDS
Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following parameters are used by the shell:

CDPATH
The search path for the *cd* command.

COLUMNS
If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing **select** lists.

EDITOR
If the value of this variable ends in *emacs*, *gmacs*, or *vi* and the **VISUAL** variable is not set, then the corresponding option (see Special Command **set** below) will be turned on.

ENV
If this parameter is set, then parameter substitution is performed on the value to generate the pathname of the script that will be executed when the *shell* is invoked. (See *Invocation* below.) This file is typically used for *alias* and *function* definitions.

FCEDIT
The default editor name for the **fc** command.

FPATH
The search path for function definitions. This path is searched when a function with the **-u** attribute is referenced and when a command is not found. If an executable file is found, then it is read and executed in the current environment.

IFS
Internal field separators, normally **space**, **tab**, and **new-line** that is used to separate command words which result from command or parameter substitution and for separating words with the special command **read**. The first character of the **IFS** parameter is used to separate arguments for the **"$*"** substitution (See *Quoting* below).

HISTFILE
If this parameter is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See *Command re-entry* below.)

HISTSIZE
If this parameter is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.

HOME
The default argument (home directory) for the **cd** command.

LINES
If this variable is set, the value is used to determine the column length for printing **select** lists. Select lists will print vertically until about two-thirds of **LINES** lines are filled.

MAIL
If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, then the shell informs the user of arrival of mail in the specified file.

MAILCHECK
This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds. When the time has elapsed the shell will check before issuing the next prompt.

MAILPATH
A colon (**:**) separated list of file names. If this parameter is set then the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a **?** and a message that will be printed. The message will undergo parameter substitution with the parameter, **$_** defined as the name of the file that has changed. The default message is *you have mail in $_*.

PATH
The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rksh* (except in *.profile*).

PS1
The value of this parameter is expanded for parameter substitution to define the primary prompt string which by default is "**$** ". The character **!** in the primary prompt string is replaced by the *command* number (see *Command Re-entry* below).

PS2
Secondary prompt string, by default "**>** ".

PS3
Selection prompt string used within a **select** loop, by default "**#?** ".

PS4
The value of this parameter is expanded for parameter substitution and precedes each line of an execution trace. If omitted, the execution trace prompt is "**+** ".

SHELL
The pathname of the *shell* is kept in the environment. At invocation, if the basename of this variable matches the pattern ***r*sh**, then the shell becomes restricted.

TMOUT
If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the **PS1** prompt. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)

VISUAL
If the value of this variable ends in *emacs*, *gmacs*, or *vi* then the corresponding option (see Special Command set below) will be turned on.

The shell gives default values to **PATH, PS1, PS2, MAILCHECK, TMOUT** and **IFS**, while **HOME, SHELL ENV** and **MAIL** are not set at all by the shell (although **HOME** *is* set by *login*(M)). On some systems **MAIL** and **SHELL** are also set by *login*(M)).

Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [unless the **-f** option has been set. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with lexicographically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a *pattern* is used for file name generation, the character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly. In other instances of pattern matching the / and . are not treated specially.

***** Matches any string, including the null string.
? Matches any single character.
[...]
Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "! " then any character not enclosed is matched. A - can be included in the character set by putting it as the first or last character.

A *pattern-list* is a list of one or more patterns separated by each other with a |. Composite patterns can be formed with one or more of the following:

?(*pattern-list*)
: Optionally matches any one of the given patterns.

*(*pattern-list*)
: Matches zero or more occurrences of the given patterns.

+(*pattern-list*)
: Matches one or more occurrences of the given patterns.

@(*pattern-list*)
: Matches exactly one of the given patterns.

!(*pattern-list*)
: Matches anything, except one of the given patterns.

Quoting

Each of the *metacharacters* listed above (See *Definitions* above) has a special meaning to the shell and causes termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (´´), are quoted. A single quote cannot appear within single quotes. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, `, ", and $. The meaning of **$*** and **$@** is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, **"$*"** is equivalent to **"$1*d*$2*d*..."**, where *d* is the first character of the **IFS** parameter, whereas **"$@"** is equivalent to **"$1" "$2" ....** Inside grave quote marks (``) \ quotes the characters \, `, and $. If the grave quotes occur within double quotes then \ also quotes the character ".

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command **let**. Evaluations are performed using *long* arithmetic. Constants are of the form [*base#*]*n* where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted then base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression of the C language. All the integral operators, other than ++, --, ?:, and , are supported. Named parameters can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a named parameter is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a *named parameter* can be specified with the **-i** option of the **typeset** special command. Arithmetic evaluation is performed on the value of each assignment to a named parameter with the **-i** attribute. If you do not specify an arithmetic base, the first assignment to the parameter determines the arithmetic base. This base is used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the **let** command is provided. For any command which begins with a ((, all the characters until a matching)) are treated as a quoted expression. More precisely, ((...)) is equivalent to **let** "...".

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Conditional Expressions

A *conditional expression* is used with the [[compound command to test attributes of files and to compare strings. Word splitting and file name generation are not performed on the words between [[and]]. Each expression can be constructed from one or more of the following unary or binary expressions:

-a *file*
: True, if *file* exists.

-b *file*
: True, if *file* exists and is a block special file.

-c *file*
: True, if *file* exists and is a character special file.

-d *file*
: True, if *file* exists and is a directory.

-f *file*
: True, if *file* exists and is an ordinary file.

-g *file*
: True, if *file* exists and is has its setgid bit set.

-k *file*
: True, if *file* exists and is has its sticky bit set.

-n *string*
: True, if length of *string* is non-zero.

-o *option*
: True, if option named *option* is on.

-p *file*
: True, if *file* exists and is a fifo special file or a pipe.

-r *file*
: True, if *file* exists and is readable by current process.

-s *file*
True, if *file* exists and has size greater than zero.

-t *fildes*
True, if file descriptor number *fildes* is open and associated with a terminal device.

-u *file*
True, if *file* exists and is has its setuid bit set.

-w *file*
True, if *file* exists and is writable by current process.

-x *file*
True, if *file* exists and is executable by current process. If *file* exists and is a directory, then the current process has permission to search in the directory.

-z *string*
True, if length of *string* is zero.

-O *file*
True, if *file* exists and is owned by the effective user id of this process.

-G *file*
True, if *file* exists and its group matches the effective group id of this process.

file1 **-nt** *file2*
True, if *file1* exists and is newer than *file2*.

file1 **-ot** *file2*
True, if *file1* exists and is older than *file2*.

file1 **-ef** *file2*
True, if *file1* and *file2* exist and refer to the same file.

string = *pattern*
True, if *string* matches *pattern*.

string != *pattern*
True, if *string* does not match *pattern*.

string1 < *string2*
True, if *string1* comes before *string2* based on ASCII value of their characters.

string1 > *string2*
True, if *string1* comes after *string2* based on ASCII value of their characters.

exp1 **-eq** *exp2*
True, if *exp1* is equal to *exp2*.

exp1 **-ne** *exp2*
True, if *exp1* is not equal to *exp2*.

exp1 **-lt** *exp2*
True, if *exp1* is less than *exp2*.

exp1 **-gt** *exp2*
True, if *exp1* is greater than *exp2*.

exp1 **-le** *exp2*
True, if *exp1* is less than or equal to *exp2*.

exp1 **-ge** *exp2*
True, if *exp1* is greater than or equal to *exp2*.

In each of the above expressions, if *file* is of the form **/dev/fd/***n*, where *n* is an integer, then the test applied to the open file whose descriptor number is *n*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

(*expression*)
True, if *expression* is true. Used to group expressions.

! *expression*
True if *expression* is false.

expression1 **&&** *expression2*
True, if *expression1* and *expression2* are both true.

expression1 **| |** *expression2*
True, if either *expression1* or *expression2* is true.

Spelling Checker

By default, the shell checks spelling whenever you use *cd* to change directories. For example, if you change to a different directory using *cd* and misspell the directory name, the shell responds with an alternative spelling of an existing directory. Enter "y" and press RETURN (or just press RETURN) to change to the offered directory. If the offered spelling is incorrect, enter "n", then retype the command line. In this example the **ksh** response is boldfaced:

```
# cd /usr/spol/uucp
/usr/spool/uucp? y
ok
```

The spell check feature is controlled by the CDSPELL environment variable. The default value of CDSPELL is set to the string "cdspell" whenever a *ksh* session is run. A user can change it to any value, including the null string, but the value is immaterial, if CDSPELL is set to any value, the spell check feature is engaged.

To disable the spelling checker, enter the following at the *ksh* prompt :

®.DE
When the user does a **set** at the *ksh* prompt,
CDSPELL is not listed if the **unset** was successful.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<*word* — Use file *word* as standard input (file descriptor 0).

>*word* — Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created. If the file exists, and the **noclobber** option is on, this causes an error; otherwise, it is truncated to zero length.

>|*word* — Sames as >, except that it overrides the **noclobber** option.

>>*word* — Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<>*word* — Open file *word* for reading and writing as standard input.

<<[-]*word* — The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\new-line** is ignored, and \ must be used to quote the characters \, $, `, and the first character of *word*. If - is appended to <<, then all leading tabs are stripped from *word* and from the document.

<&*digit* — The standard input is duplicated from file descriptor *digit* (see *dup*(S)). Similarly for the standard output using >& *digit*.

<&- — The standard input is closed. Similarly for the standard output using **>&-**.

<&p — The input from the co-process is moved to standard input.

>&p — The output to the co-process is moved to standard output.

If one of the above is preceded by a digit, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

... 1>*fname* 2>&1

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *fname*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by **&** and job control is not active, then the default standard input for the command is the empty file **/dev/null.** Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see *environ*(M)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value and marking it *export* . Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones, using the **export** or **typeset -x** commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in **export** or **typeset -x** commands.

The environment for any *simple-command* or function may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument is a word of the form *identifier=value*. Thus:

TERM=450 cmd args and
(export TERM; TERM=450; cmd args)

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* parameter assignment arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c:**

```
echo a=b c
set -k
echo a=b c
```

This feature is intended for use with scripts written for early versions of the shell and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions

The **function** reserved word, described in the *Commands* section above, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See *Execution* below).

Functions execute in the same process as the caller and share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on **EXIT** set inside a function is executed after the function completes in the environment of the caller. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the **-f** or **+f** option of the **typeset** special command. The text of functions will also be listed with **-f**. Function can be undefined with the **-f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **-xf** option of the **typeset** command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be specified in the **ENV** file with the **-xf** option of **typeset**

Jobs

If the **monitor** option of the **set** command is turned on, an interactive shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line which looks like:

[1] 1234

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

This paragraph and the next require features that are not in all versions of the UNIX operating system and may not apply. If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command "stty tostop". If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. A job can be referred to by the process id of any process of the job or by one of the following:

%number
The job with the given number.

%string
Any job whose command line begins with *string*.

%?string
Any job whose command line contains *string*.

%%
Current job.

%+
Equivalent to *%%*.

%-
Previous job.

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

When the monitor mode is on, each background job that completes triggers any trap set for **CHLD**.

When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped(running) jobs.' You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by & and job **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. When the *function* completes or issues a **return**, the positional parameter list is restored and any trap set on EXIT within the function is executed. The value of a *function* is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a *special command* or a user defined *function*, a process is created and an attempt is made to execute the command via *exec*(S).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **/bin:/usr/bin:** (specifying **/bin**, **/usr/bin**, and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and named parameters are removed in this case. If the shell command file doesn't have read permission, or if the *setuid* and/or *setgid* bits are set on the file, then the shell executes an agent whose job it is to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified.

If you do not specify an editor program as an argument to **fc** then the value of the parameter **FCEDIT** is used. If **FCEDIT** is not defined then **/bin/ed** is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name **-** is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if **r** is aliased to ′**fc -e -**′ then typing ‘**r bad=good c**’ will re-execute the most recent command which starts with the letter **c**, replacing the first occurrence of the string **bad** with the string **good**.

In-line Editing Options

Normally, each command line entered from a terminal device is simply typed followed by a new-line (‘RETURN’ or ‘LINE FEED’). If either the **emacs**, **gmacs**, or **vi** option is active, the user can edit the command line. To be in either of these edit modes **set** the corresponding option. An editing option is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept ‘RETURN’ as carriage return without line feed and that a space (‘ ’) must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to ‘space’. Hewlett-Packard series 2621 terminal users should set the straps to ‘bcGHxZ etX’.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of **COLUMNS** if it is defined, otherwise 80. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > (<, *) if the line extends on the right (left, both) side(s) of the window.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading ^ in the string restricts the match to begin at the first character in the line.

Emacs Editing Mode

This mode is entered by enabling either the *emacs* or *gmacs* option. The only difference between these two modes is the way they handle **^T**. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, **^F** is the notation for control **F**. This is entered by depressing ‘f’ while holding down the ‘CTRL’ (control) key. The ‘SHIFT’ key is *not* depressed. (The notation ^? indicates the DEL (delete) key.)

The notation for escape sequences is **M-** followed by a character. For example, **M-f** (pronounced Meta f) is entered by depressing ESC (ascii **033**) followed by 'f'. (**M-F** would be the notation for ESC followed by 'SHIFT' (capital) 'F'.)

All edit commands operate from any place on the line (not just at the beginning). Neither the "RETURN" nor the "LINE FEED" key is entered after edit commands except when noted.

^F	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The emacs editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]*char*	Move cursor forward to character *char* on current line.
M-^]*char*	Move cursor back to character *char* on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the *stty*(C) command, usually **^H** or **#**.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is ^? (DEL, the default) then this command will not work).
^T	Transpose current character with next character in *emacs* mode. Transpose two previous characters in *gmacs* mode.
^C	Capitalize current character.
M-c	Capitalize current word.
M-l	Change the current word to lower case.
^K	Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less than the current cursor position, then delete from given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from cursor up to given cursor position.
^W	Kill from the cursor to the mark.
M-p	Push the region from the cursor to the mark on the stack.
kill	(User defined kill character as defined by the stty command, usually **^G** or **@**.) Kill the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).

^Y	Restore last item removed from line. (Yank item back to the line.)
^L	Line feed and print current line.
^@	(Null character) Set mark.
M-*space*	(Meta space) Set mark.
^J	(New line) Execute the current line.
^M	(Return) Execute the current line.
eof	End-of-file character, normally **^D**, is processed as an End-of-file only if the current line is null.
^P	Fetch previous command. Each time **^P** is entered the previous command back in time is accessed. Moves back one line when not on the first line of a multi-line command.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.
^N	Fetch next command line. Each time **^N** is entered the next command line forward in time is accessed.
^R*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *String* is terminated by a "RETURN" or "NEW LINE". If string is preceded by a ^, the matched line must begin with *string*. If *string* is omitted, then the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.
^O	Operate - Execute the current line and fetch the next line relative to current line from the history file.
M-*digits*	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are **^F**, **^B**, *erase*, **^C**, **^D**, **^K**, **^R**, **^P**, **^N**, **^]**, **M-.**, **M-^]**, **M-_**, **M-b**, **M-c**, **M-d**, **M-f**, **M-h** **M-l** and **M-^H**.
M-*letter*	Soft-key - Your alias list is searched for an alias by the name _*letter* and if an alias of this name is defined, its value will be inserted on the input queue. The *letter* must not be one of the above meta-functions. **M-]***letter* Soft-key - Your alias list is searched for an alias by the name __*letter* and if an alias of this name is defined, its value will be inserted on the input queue. The can be used to program functions keys on many terminals.
M-.	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_	Same as **M-.**.
M-*	Attempt file name generation on the current word. An asterisk is appended if the word doesn't match any file or contain any special pattern characters.

M-ESC File name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

M-= List files matching current word pattern if an asterisk were appended.

^U Multiply parameter of next command by 4.

\ Escape next character. Editing characters, the user's erase, kill and interrupt (normally ^?) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).

^V Display version of the shell.

M-# Insert a # at the beginning of the line and execute it. This causes a comment to be inserted in the history file.

Vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the *input* mode. To edit, the user enters *control* mode by typing ESC (**033**) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* prior to the command.

When in vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the option **viraw** is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase (User defined erase character as defined by the stty command, usually **^H** or #.) Delete previous character.

^W Delete the previous blank separated word.

^D Terminate the shell.

^V Escape next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a **^V**. The **^V** removes the next character's editing features (if any).

\ Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands will move the cursor.

[*count*]**l**	Cursor forward (right) one character.
[*count*]w	Cursor forward one alpha-numeric word.
[*count*]**W**	Cursor to the beginning of the next word that follows a blank.
[*count*]**e**	Cursor to end of word.
[*count*]**E**	Cursor to end of the current blank delimited word.
[*count*]**h**	Cursor backward (left) one character.
[*count*]**b**	Cursor backward one word.
[*count*]**B**	Cursor to preceding blank separated word.
[*count*]**|**	Cursor to column *count*.
[*count*]**f***c*	Find the next character *c* in the current line.
[*count*]**F***c*	Find the previous character *c* in the current line.
[*count*]**t***c*	Equivalent to **f** followed by **h**.
[*count*]**T***c*	Equivalent to **F** followed by **l**.
[*count*]**;**	Repeats *count* times, the last single character find command, **f**, **F**, **t**, or **T**.
[*count*]**,**	Reverses the last single character find command *count* times.
0	Cursor to start of line.
^	Cursor to first non-blank character in line.
$	Cursor to end of line.

Search Edit Commands

These commands access your command history.

[*count*]**k**	Fetch previous command. Each time **k** is entered the previous command back in time is accessed.
[*count*]**-**	Equivalent to **k**.
[*count*]**j**	Fetch next command. Each time **j** is entered the next command forward in time is accessed.
[*count*]**+**	Equivalent to **j**.
[*count*]**G**	The command number *count* is fetched. The default is the least recent history command.
/string	Search backward through history for a previous command containing *string*. *String* is terminated by a "RETURN" or "NEW LINE". If string is preceded by a ^, the matched line must begin with *string*. If *string* is null the previous string will be used.
?*string*	Same as / except that search will be in the forward direction.
n	Search for next match of the last pattern to / or **?** commands.
N	Search for next match of the last pattern to / or **?**, but in reverse direction. Search history for the *string* entered by the previous / command.

Text Modification Edit Commands

These commands will modify the line.

a Enter input mode and enter text after the current character.

A Append text to the end of the line. Equivalent to **$a**.

[*count*]**c***motion*
c[*count*]*motion*
Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is **c**, the entire line will be deleted and input mode entered.

C Delete the current character through the end of line and enter input mode. Equivalent to **c$**.

S Equivalent to **cc**.

D Delete the current character through the end of line. Equivalent to **d$**.

[*count*]**d***motion*
d[*count*]*motion*
Delete current character through the character that *motion* would move to. If *motion* is **d** , the entire line will be deleted.

i Enter input mode and insert text before the current character.

I Insert text before the beginning of the line. Equivalent to **0i**.

[*count*]**P** Place the previous text modification before the cursor.

[*count*]**p** Place the previous text modification after the cursor.

R Enter input mode and replace characters on the screen with characters you type overlay fashion.

[*count*]**r***c* Replace the *count* character(s) starting at the current cursor position with *c*, and advance the cursor.

[*count*]**x** Delete current character.

[*count*]**X** Delete preceding character.

[*count*]**.** Repeat the previous text modification command.

[*count*]**~** Invert the case of the *count* character(s) starting at the current cursor position and advance the cursor.

[*count*]**_** Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.

***** Causes an * to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

**** Filename completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

Other Edit Commands

Miscellaneous commands.

[*count*]y*motion*
y[*count*]*motion*
: Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.

Y Yanks from current position to end of line. Equivalent to **y$**.

u Undo the last text modifying command.

U Undo all the text modifying commands performed on the line.

[*count*]**v** Returns the command **fc -e ${VISUAL:-${EDITOR:-vi}}** *count* in the input buffer. If *count* is omitted, then the current line is used.

^L Line feed and print current line. Has effect only in control mode.

^J (New line) Execute the current line, regardless of mode.

^M (Return) Execute the current line, regardless of mode.

Sends the line after inserting a # in front of the line. Useful for causing the current line to be inserted in the history without being executed.

= List the file names that match the current word if an asterisk were appended it.

@letter Your alias list is searched for an alias by the name _*letter* and if an alias of this name is defined, its value will be inserted on the input queue for processing.

Special Commands.

The following simple-commands are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is zero. Commands that are preceded by one or two † are treated specially in the following ways:

1. Parameter assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after parameter assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a parameter assignment, are expanded with the same rules as a parameter assignment. This means that tilde substitution is performed after the = sign and word splitting and file name generation are not performed.

† : [*arg* ...]
The command only expands parameters.

† . *file* [*arg* ...]
Read the complete *file* then execute the commands. The commands are executed in the current Shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed.

†† **alias** [**-tx**] [*name*[=*value*]] ...
Alias with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **-t** flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the **-t** flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **-x** flag is used to set or print exported aliases. An exported alias is defined for scripts invoked by name. The exit status is non-zero if a *name* is given, but no value, for which no alias has been defined.

bg [*job*...]
This command is only on systems that support job control. Puts each specified *job* into the background. The current job is put in the background if *job* is not specified. See *Jobs* for a description of the format of *job*.

† **break** [*n*]
Exit from the enclosing **for while until** or **select** loop, if any. If *n* is specified then break *n* levels.

† **continue** [*n*]
Resume the next iteration of the enclosing **for while until** or **select** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*]
cd *old new*
This command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is - the directory is changed to the previous directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or

between the colon delimiters anywhere else in the path list. If *arg* begins with a / then the search path is not used. Otherwise, each directory in the path is searched for *arg*.
The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.
The **cd** command may not be executed by *rksh*.

echo [*arg* ...]
See *echo*(C) for usage and description.

† **eval** [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

† **exec** [*arg* ...]
If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

† **exit** [*n*]
Causes the shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed. An end-of-file will also cause the shell to exit except for a shell which has the *ignoreeof* option (See **set** below) turned on.

†† **export** [*name*[=*value*]] ...
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands.

fc [**-e** *ename*] [**-nlr**] [*first* [*last*]]
fc -e - [*old=new*] [*command*]
In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag **-l**, is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the parameter **FCEDIT** (default /bin/ed) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and -16 for listing. The flag **-r** reverses the order of the commands and the flag **-n** suppresses command numbers when listing. In the second form the *command* is re-executed after the substitution *old=new* is performed.

fg [*job*...]

This command is only on systems that support job control. Each *job* specified is brought to the foreground. Otherwise, the current job is brought into the foreground. See *Jobs* for a description of the format of *job*.

getopts *optstring name* [*arg* ...]

Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a + or a -. An option not beginning with + or - or the argument -- ends the options. *optstring* contains the letters that *getopts* recognizes. If a letter is followed by a :, that option is expected to have an argument. The options can be separated from the argument by blanks. **getopts** places the next option letter it finds inside variable *name* each time it is invoked with a + prepended when *arg* begins with a +. The index of the next *arg* is stored in **OPTIND**. The option argument, if any, gets stored in **OPTARG**.
A leading : in *optstring* causes **getopts** to store the letter of an invalid option in **OPTARG**, and to set *name* to **?** for an unknown option and to : when a required option is missing. Otherwise, **getopts** prints an error message. The exit status is non-zero when there are no more options.

jobs [**-lnp**] [*job* ...]

Lists information about each given job; or all active jobs if *job* is omitted. The **-l** flag lists process ids in addition to the normal information. The **-n** flag only displays jobs that have stopped or exited since last notified. The **-p** flag causes only the process group to be listed. See *Jobs* for a description of the format of *job*.

kill [*-sig*] *job* ...
kill -l

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in **/usr/include/signal.h**, stripped of the prefix "SIG"). If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument *job* can the process id of a process that is not a member of one of the active jobs. See *Jobs* for a description of the format of *job*. In the second form, **kill -l**, the signal numbers and names are listed.

let *arg* ...

Each *arg* is a separate *arithmetic expression* to be evaluated. See *Arithmetic Evaluation* above, for a description of arithmetic expression evaluation.
The exit status is 0 if the value of the last expression is non-zero, and 1 otherwise.

† **newgrp** [*arg* ...]

Equivalent to **exec /bin/newgrp** *arg* ....

print [**-Rnprsu**[*n*]] [*arg* ...]
The shell output mechanism. With no flags or with flag **-** or **- -** the arguments are printed on standard output as described by *echo*(C). In raw mode, **-R** or **-r**, the escape conventions of *echo* are ignored. The **-R** option will print all subsequent arguments and options other than **-n**. The **-p** option causes the arguments to be written onto the pipe of the process spawned with **|&** instead of standard output. The **-s** option causes the arguments to be written onto the history file instead of standard output. The **-u** flag can be used to specify a one digit file descriptor unit number **n** on which the output will be placed. The default is 1. If the flag **-n** is used, no **newline** is added to the output.

pwd
Equivalent to **print -r - $PWD**

read [**-prsu**[*n*]] [*name?prompt*] [*name* ...]
The shell input mechanism. One line is read and is broken up into fields using the characters in **IFS** as separators. In raw mode, **-r,** a \ at the end of a line does not signify line continuation. The first field is assigned to the first *name*, the second field to the second *name*, etc., with leftover fields assigned to the last *name*. The **-p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **-s** flag is present, the input will be saved as a command in the history file. The flag **-u** can be used to specify a one digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is 0. If *name* is omitted then **REPLY** is used as the default *name*. The exit status is 0 unless an end-of-file is encountered. An end-of-file with the **-p** option causes cleanup for this process so that another can be spawned. If the first argument contains a ?, the remainder of this word is used as a *prompt* on standard error when the shell is interactive. The exit status is 0 unless an end-of-file is encountered.

†† **readonly** [*name*[*=value*]] ...
The given *names* are marked readonly and these names cannot be changed by subsequent assignment.

† **return** [*n*]
Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted then the return status is that of the last command executed. If **return** is invoked while not in a *function* or a **.** script, then it is the same as an **exit**.

set [**±aefhkmnopstuvx**] [**±o** *option*]... [**±A** *name*] [*arg* ...]
The flags for this command have meaning as follows:

-A Array assignment. Unset the variable *name* and assign values sequentially from the list *arg*. If +A is used, the variable *name* is not unset first.

-a All subsequent parameters that are defined are automatically exported.

-e If a command has a non-zero exit status, execute the **ERR** trap, if set, and exit. This mode is disabled while reading profiles.

-f Disables file name generation.

-h Each command becomes a tracked alias when first encountered.

-k All parameter assignment arguments are placed in the environment for a command, not just those that precede the command name.

-m Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.

-n Read commands and check them for syntax errors, but do not execute them. Ignored for interactive shells.

-o The following argument can be one of the following option names:

allexport	Same as **-a**.
errexit	Same as **-e**.
bgnice	All background jobs are run at a lower priority. This is the default mode.
emacs	Puts you in an *emacs* style in-line editor for command entry.
gmacs	Puts you in a *gmacs* style in-line editor for command entry.
ignoreeof	The shell will not exit on end-of-file. The command **exit** must be used.
keyword	Same as **-k**.
markdirs	All directory names resulting from file name generation have a trailing / appended.
monitor	Same as **-m**.
noclobber	Prevents redirection > from truncating existing files. Require >| to truncate a file when turned on.
noexec	Same as **-n**.
noglob	Same as **-f**.
nolog	Do not save function definitions in history file.
nounset	Same as **-u**.
privileged	Same as **-p**.
verbose	Same as **-v**.
trackall	Same as **-h**.
vi	Puts you in insert mode of a *vi* style in-line editor until you hit escape character **033**. This puts you in move mode. A return sends the line.

viraw Each character is processed as it is typed in *vi* mode.

xtrace Same as **-x**.

If no option name is supplied then the current option settings are printed.

-p Disables processing of the **$HOME/.profile** file and uses the file **/etc/suid_profile** instead of the **ENV** file. This mode is on whenever the effective uid (gid) is not equal to the real uid (gid). Turning this off causes the effective uid and gid to be set to the real uid and gid.

-s Sort the positional parameters lexicographically.

-t Exit after reading and executing one command.

-u Treat unset parameters as an error when substituting.

-v Print shell input lines as they are read.

-x Print commands and their arguments as they are executed.

- Turns off -x and -v flags and stops examining arguments for flags.

-- Do not change any of the flags; useful in setting **$1** to a value beginning with -. If no arguments follow this flag then the positional parameters are unset.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $-. Unless -A is specified, the remaining arguments are positional parameters and are assigned, in order, to **$1 $2** .... If no arguments are given then the names and values of all named parameters are printed on the standard output. If the only argument is +, the names of all named parameters are printed.

† **shift** [*n*]

The positional parameters from $*n*+1 ... are renamed **$1** ... , default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

† **times**

Print the accumulated user and system times for the shell and for processes run from the shell.

† **trap** [*arg*] [*sig*] ...

arg is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is omitted or is -, then all trap(s) *sig* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *sig* is **ERR** then *arg* will be executed whenever a command has a non-zero exit status. *sig* is **DEBUG** then *arg* will be executed after each command. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

†† **typeset** [**±HLRZfilrtux**[*n*]] [*name*[=*value*]] ...
Sets attributes and values for shell parameters. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

-H This flag provides UNIX system to host-name file mapping on non-UNIX system machines.

-L Left justify and remove leading blanks from *value*. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. When the parameter is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the **-Z** flag is also set. The **-R** flag is turned off.

-R Right justify and fill with leading blanks. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left filled with blanks or truncated from the end if the parameter is reassigned. The **L** flag is turned off.

-Z Right justify and fill with leading zeros if the first non-blank character is a digit and the **-L** flag has not been set. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment.

-f The names refer to function names rather than parameter names. No assignments can be made and the only other valid flags are **-t**, **-u** and **-x**. The flag **-t** turns on execution tracing for this function. The flag **-u** causes this function to be marked undefined. The **FPATH** variable will be searched to find the function definition when the function is referenced. The flag **-x** allows the function definition to remain in effect across shell procedures invoked by name.

-i Parameter is an integer. This makes arithmetic faster. If *n* is non-zero it defines the output arithmetic base, otherwise the first assignment determines the output base.

-l All upper-case characters converted to lower-case. The upper-case flag, **-u** is turned off.

-r The given *names* are marked readonly and these names cannot be changed by subsequent assignment.

-t Tags the named parameters. Tags are user definable and have no special meaning to the shell.

-u All lower-case characters are converted to upper-case characters. The lower-case flag, **-l** is turned off.

-x The given *names* are marked for automatic export to the *environment* of subsequently-executed commands.

Using + rather than - causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of *names* (and optionally the *values*) of the *parameters* which have these flags set is printed. (Using + rather than - keeps the values from being printed.) If no *name*s and flags are given, the *names* and -*attributes* of all *parameters* are printed.

ulimit [**-HS**] [*limit*]
The number of 512-byte blocks on files written by child processes (files of any size may be read). The limit is set when *limit* is specifed. The value of *limit* can be a number or the value **unlimited.** The **H** and **S** flags specify whether the hard limit or the soft limit is set. A hard limit cannot be increased once it is set. A soft limit can be increased up to the value of the hard limit. If neither the **H** or **S** options is specified, the limit applies to both. The current limit is printed when *limit* is omitted. In this case the soft limit is printed unless **H** is specified.

umask [*mask*]
The user file-creation mask is set to *mask* (see *umask*(C)). *mask* can either be an octal number or a symbolic value as described in *chmod*(C). If a symbolic value is given, the new umask value is the complement of the result of applying *mask* to the complement of the previous umask value. If *mask* is omitted, the current value of the mask is printed.

unalias *name* ...
The parameters given by the list of *name*s are removed from the *alias* list.

unset [**-f**] *name* ...
The parameters given by the list of *name*s are unassigned, i. e., their values and attributes are erased. Readonly variables cannot be unset. If the flag, **-f**, is set, then the names refer to *function* names. Unsetting **ERRNO, LINENO, MAILCHECK, OPTARG, OPTIND, RANDOM, SECONDS, TMOUT,** and _ causes removes their special meaning even if they are subsequently assigned to.

† **wait** [*job*]
Wait for the specified *job* and report its termination status. If *job* is not given then all currently active child processes are waited for. The exit status from this command is that of the process waited for. See *Jobs* for a description of the format of *job*.

whence [**-pv**] *name* ...
For each *name*, indicate how it would be interpreted if used as a command name.
The flag, **-v**, produces a more verbose report.
The flag, **-p**, does a path search for *name* even if name is an alias, a function, or a reserved word.

Invocation.

If the shell is invoked by *exec*(S), and the first character of argument zero (**$0**) is -, then the shell is assumed to be a *login* shell and commands are read from **/etc/profile** and then from either **.profile** in the current directory or **$HOME/.profile**, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV** if the file exists. If the **-s** flag is not present and *arg* is, then a path search is performed on the first *arg* to determine the name of the script to execute. The script *arg* must have read permission and any *setuid* and *getgid* settings will be ignored. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

-c *string* If the **-c** flag is present then commands are read from *string*.

-s If the **-s** flag is present or if no arguments remain then commands are read from the standard input. Shell output, except for the output of the *Special commands* listed above, is written to file descriptor 2.

-i If the **-i** flag is present or if the shell input and output are attached to a terminal (as told by *ioctl*(S)) then this shell is *interactive*. In this case TERM is ignored (so that **kill 0** does not kill an interactive shell) and INTR is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

-r If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

rksh Only.

rksh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rksh* are identical to those of *ksh*, except that the following are disallowed:
changing directory (see *cd*(C)),
setting the value of **SHELL**, **ENV**, or **PATH**,
specifying path or command names containing /,
redirecting output (>, >| , <> , and >>).

The restrictions above are enforced after **.profile** and the **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, *rksh* invokes *ksh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rksh*. Some systems also provide a restricted editor *red*.

Diagnostics

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above). If the shell is being used non-interactively then execution of the shell file is abandoned. Run time errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, then the line number is also printed in square brackets ([]) after the command or function name.

Files

/etc/passwd
/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh*
/dev/null

See Also

cat(C), cd(C), chmod(C), cut(C), echo(C), env(C), newgrp(C), stty(C), test(C), umask(C), vi(C), dup(S), exec(S), fork(S), ioctl(S), lseek(S), paste(C), pipe(S), signal(S), umask(S), ulimit(S), wait(S), rand(S), a.out(F), profile(M), environ(M).

Notes

If a command which is a *tracked alias* is executed, and then a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **-t** option of the **alias** command to correct this situation.

Some very old shell scripts contain a ^ as a synonym for the pipe character. **|**.

Using the **fc** built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command **.** *file* reads the whole file before any commands are executed. Therefore, **alias** and **unalias** commands in the file will not apply to any functions defined in the file.

Traps are not processed while a job is waiting for a foreground process. Thus, a trap on **CHLD** won't be executed until the forground job terminates.

l

lists information about contents of directory

Syntax

l [**-ACFRabcdfgilnopqrstu**] name ...

Description

For each directory argument, *l* lists the contents of the directory. For each *name*, *l* repeats its name and other requested information. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. Information is listed in the format of the "ls -l" command, which is identical to the *l* command. This format and all provided switches are described in *ls*(C) and *lc*(C), to which should you should refer for a complete discussion of the capabilities of *l*.

Files

/etc/passwd	Contains user IDs
/etc/group	Contains group IDs

Notes

Newline and tab are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

last

indicate last logins of users and teletypes

Syntax

last [**-h**] [**-n** limit] [**-t** tty] [**-w** wtmpfile] [name]

Description

last checks the *wtmp* file, which records all logins and logouts for information about a user, a serial line or any group of users and lines. Arguments specify a user name and/or tty.

last -t 01 root

would list all "root" sessions as well as all sessions on **/dev/tty01.** *last* prints the sessions of the specified users and ttys, including login name, the line used, the device name, the process ID, plus start time and elapsed time.

last with no arguments prints a record of all logins and logouts, in reverse chronological order.

The options behave as follows:

-h no header.

-n *limit*
limits the report to n lines.

-t *line*
specifies the tty.

-w*wtmpfile*
uses *wtmpfile* instead of **/etc/wtmp**.

Files

/etc/wtmp	login data base

See Also

finger(C), utmp(M), accton(ADM), acctcom(ADM), acct(F)

layers

layer multiplexer for windowing terminals

Syntax

layers [**-s**] [**-t**] [**-d**] [**-p**] [**-f** file] [layersys-prgm]

Description

The *layers* command manages asynchronous windows [see *layers*(M)] on a windowing terminal. Upon invocation, *layers* finds an unused *xt*(HW) channel group and associates it with the terminal line on its standard output. It then waits for commands from the terminal.

Command-line options:

-s Reports protocol statistics on standard error at the end of the session after you exit from *layers*. The statistics may be printed during a session by invoking the program *xts*(ADM).

-t Turns on *xt*(HW) driver packet tracing, and produces a trace dump on standard error at the end of the session after you exit from *layers*. The trace dump may be printed during a session by invoking the program *xtt*(ADM).

-d If a firmware patch has been downloaded, prints out the sizes of the text, data, and bss portions of the firmware patch on standard error.

-p If a firmware patch has been downloaded, prints the downloading protocol statistics and a trace on standard error.

-f *file* Starts *layers* with an initial configuration specified by *file*. Each line of the file represents a layer to be created, and has the following format:

origin_x origin_y corner_x corner_y command_list

The coordinates specify the size and position of the layer on the screen in the terminal's coordinate system. If all four are *0*, the user must define the layer interactively. *command_list*, a list of one or more commands, must be provided. It is executed in the new layer using the user's shell (by executing: **$SHELL -i -c** "*command_list*"). This means that the last command should invoke a shell, such as **/bin/sh.** (If the last command is not a shell, then, when the last command has completed, the layer will not be functional.)

layersys-prgm
A file containing a firmware patch that the *layers* command downloads to the terminal before layers are created and *command_list* is executed.

Each layer is in most ways functionally identical to a separate terminal. Characters typed on the keyboard are sent to the standard input of the UNIX system process attached to the current layer (called the host process), and characters written on the standard output by the host process appear in that layer. When a layer is created, a separate shell is established and bound to the layer. If the environment variable **SHELL** is set, the user will get that shell, otherwise, **/bin/sh** will be used. In order to enable communications with other users via *write*(C), *layers* invokes the command *relogin*(ADM) when the first layer is created. *relogin*(ADM) will reassign that layer as the user's logged-in terminal. An alternative layer can be designated by using *relogin*(ADM) directly. *layers* will restore the original assignment on termination.

Layers are created, deleted, reshaped, and otherwise manipulated in a terminal-dependent manner. For instance, the AT&T TELETYPE 5620 DMD terminal provides a mouse-activated pop-up menu of layer operations. The method of ending a *layers* session is also defined by the terminal.

Example

layers -f startup

where **startup** contains

```
8 8 700 200 date ; pwd ; exec $SHELL
8 300 780 850 exec $SHELL
```

Notes

The *xt*(HW) driver supports an alternate data transmission scheme known as ENCODING MODE. This mode makes *layers* operation possible even over data links which intercept control characters or do not transmit 8-bit characters. ENCODING MODE is selected either by setting a configuration option on your windowing terminal or by setting the environment variable **DMDLOAD** to the value *hex* before running *layers*:

export DMDLOAD; DMDLOAD=hex

If, after executing **layers -f** *file*, the terminal does not respond in one or more of the layers, often the last command in the *command-list* for that layer did not invoke a shell.

When invoking *layers* with the **-s**, **-t**, **-d**, or **-p** options, it is best to redirect standard error to another file to save the statistics and tracing output (e.g., **layers -s 2>stats**); otherwise all or some of the output may be lost.

Files

/dev/xt??[0-7]
/usr/lib/layersys/lsys.8;7;3
/usr/lib/layersys/lsys.8;?;?

See Also

relogin(ADM), sh(C), write(C), wtinit(ADM), xts(ADM), xtt(ADM), xt(HW), libwindows(S), layers(M)

lc

lists directory contents in columns

Syntax

lc [**-1ACFRabcdfgilmnopqrstux**] *name* ...

Description

lc lists the contents of files and directories, in columns. If *name* is a directory name, *lc* lists the contents of the directory; if *name* is a filename, *lc* repeats the filename and any other information requested. Output is given in columns and sorted alphabetically. If no argument is given, the current directory is listed. If several arguments are given, they are sorted alphabetically, but file arguments appear before directories.

Files that are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. A stream output format is available in which files are listed across the page, separated by commas. The **-m** option enables this format.

The options are:

-1 Forces an output format with one entry per line.

-A If not the root directory, this option displays all files that begin with "." (except "." and ".." themselves). Otherwise, files are displayed normally.

-C Forces columnar output, even if redirected to a file.

-F Causes directories to be marked with a trailing "/" and executable files to be marked with a trailing "*".

-R Recursively lists subdirectories.

-a Lists all entries; "." and ".." are not suppressed.

-b Forces printing of nongraphic characters in the *ddd* notation, in octal.

-c Sorts by time of file creation, for use with **-t** option.

-d If the argument is a directory, lists only its name, not its contents (mostly used with **-l** to get status on directory).

-f Forces each argument to be interpreted as a directory and lists the name found in each slot. This option turns off **-l, -t, -s,** and **-r,** and turns on **-a.** The order is the order in which entries appear in the directory.

-g The same as -l, except that the owner is not printed.

-i Prints inode number in first column of the report for each file listed.

-l Lists in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file. If the file is a special file, the size field contains the major and minor device numbers instead.

-m
Forces stream output format.

-n Same as the -l switch, but the owner's user ID appears instead of the owner's name. If used in conjunction with the **-g** switch, the owner's group ID appears instead of the group name.

-o The same as -l, except that the group is not printed.

-p Pad output with spaces.

-q Forces printing of nongraphic characters in filenames as the character "?".

-r Reverses the order of sort to get reverse alphabetic or oldest first as appropriate.

-s Gives size in 512-byte blocks, including indirect blocks for each entry.

-t Sorts by time modified (latest first) instead of by name, as is normal.

-u Uses time of last access instead of last modification for sorting (**-t**) or printing (**-l**).

-x Forces columnar printing to be sorted across rather than down the page.

The following are alternate invocations of the **lc** command:

lf Produces the same output as **lc** -**F**.

lr Produces the same output as **lc** -**R**.

lx Produces the same output as **lc -x**.

The mode printed under the **-l** option contains 11 characters. The first character is:

- If the entry is a plain file

d If the entry is a directory

b If the entry is a block-type special file

c If the entry is a character-type special file

p If the entry is a named pipe

s If the entry is a semaphore

m If the entry is shared data (memory)

The next 9 characters are interpreted as 3 sets of 3 bits each. The first set refers to owner permissions; the next to permissions of others in the same user-group; and the last to all others. Within each set, the 3 characters indicate permission to read, to write, or to execute the file as a program, respectively. For a directory, ''execute'' permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r If the file is readable

w If the file is writable

x If the file is executable

- If the indicated permission is not granted

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise the user-execute permission character is given as **s** if the file has set-user-ID mode.

The last character of the mode (normally ''x'' or ''-'') is **t** if the 1000 bit of the mode is on. See *chmod*(C) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is displayed.

Files

/etc/passwd	To get user IDs for "lc -o"
/etc/group	To get group IDs for "lc -g"

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Newline and tab are considered printing characters in filenames. The output device is assumed to be 80 columns wide. Column width choices are poor for terminals that can tab.

This utility reports sizes in 512 byte blocks. *lc -s* will report 2 blocks used, rather than 1 block, since the file uses one system block of 1024 bytes.

line

reads one line

Syntax

line

Description

line copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on end-of-file and always prints at least a newline. It is often used within shell files to read from the user's terminal.

See Also

gets(CP), sh(C)

Standards Conformance

line is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ln

makes a link to a file

Syntax

ln [**-f**] file1 [file2 ...] target

Description

A link is a directory entry referring to a file. The same file (together with its size, all its protection information, etc) may have several links to it. There is no way to distinguish a link to a file from its original directory entry. Any changes to the file are effective independent of the name by which the file is known.

If *target* is a directory, then one or more files are linked to that directory.

If *ln* determines that the mode of *target* forbids writing, it will print the mode [see *chmod*(C)], ask for a response, and read the standard input for one line. If the line begins with **y**, the *ln* occurs, if permissible; if not, the command exits.

When the **-f** option is used or if the standard input is not a terminal, no questions are asked and the *ln* is done.

See Also

cp(C), mv(C), rm(C)

Standards Conformance

ln is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

lock

locks a user's terminal

Syntax

lock [-v] [-number]

Description

lock requests a password from the user, requests it again for verification, then locks the terminal until the password is reentered. If a *-number* is specified in the *lock* command, the terminal is automatically logged out and made available to another user after that number of minutes has passed.

This command uses the file */etc/default/lock.* This file has two entries:

```
DEFLOGOUT = number
MAXLOGOUT = number
```

DEFLOGOUT specifies the default time in minutes a terminal will remain locked before the user is logged out. This default value is overridden if the *-number* option is used on the command line. If DEFLOGOUT and *-number* are not specified, the MAXLOGOUT value is used.

MAXLOGOUT is the maximum number of minutes a user is permitted to lock a terminal. If a user attempts to lock a terminal for longer than this time, *lock* will issue a warning to the user that it is using the system maximum time limit. If DEFLOGOUT and *-number* and MAXLOGOUT are not specified, users are not logged out.

DEFLOGOUT and MAXLOGOUT are configured by the system administrator to reflect the demand for terminals at the site.

The lock may be terminated by killing the lock process. Only the superuser and the user who invoked *lock* may do so.

Options

-number Sets the time limit for lock to *number* of minutes, instead of the system default.

-v Specifies verbose operation.

Files

/etc/default/lock

Notes

The file */etc/default/lock* is shipped with the following default values:

```
DEFLOGOUT = 30
MAXLOGOUT = 60
```

logname

gets login name

Syntax

logname

Description

logname returns the user's login name as found in */etc/utmp*. If no login name is found, *logname* returns the user's user ID number.

See Also

env(C), id(C), getlogin(S), getuid(S), login(M), logname(S)

Standards Conformance

logname is conformant with:

The X/Open Portability Guide II of January 1987.

lp, cancel

send/cancel requests to lineprinter

Syntax

lp [options] *files*
lp **-i** *id printing options*
cancel [*ids*] [*printers*]

Description

The first form of the *lp* shell command arranges for the named files and associated information (collectively called a *request*) to be printed. If no file names are specified on the shell command line, the standard input is assumed. The standard input may be specified along with named *files* on the shell command line using the file name. The *files* will be printed in the order they appear on the shell command line.

The second form of *lp* is used to change the options for a request. The print request identified by the *request-id* is changed according to the printing options specified with this shell command. The printing options available are the same as those with the first form of the *lp* shell command. If *request-id* has finished printing, the change is rejected. If the *request-id* is already printing, it will be stopped and restarted from the beginning, unless the **-P** option has been given.

lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel, change, or find the status of the request. (See the section on *cancel* for details about canceling a request, the previous paragraph for an explanation of how to change a request, and *lpstat*(C) for information about checking the status of a print request.)

Sending a Print Request

The first form of the *lp* command is used to send a print request to a particular printer or group of printers.

Options to *lp* must always precede file names but may be listed in any order. The following options are available for *lp*:

-c When *lp* is invoked, copies of the *files* to be printed are made immediately. Normally, *files* will not be copied. If the **-c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **-c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

-d *dest* Prints this request using *dest* as the printer or class of printers. Under certain conditions (lack of printer availability, capabilities of printers, and so on), requests for specific destinations may not be accepted [see *accept*(ADM) and *lpstat*(C)]. By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems [see *lpstat*(C)].

-f *form-name* [**-d any**]
Prints the request on the form *form-name*. The LP print service ensures that the form is mounted on the printer. If *form-name* is requested with a printer destination that cannot support the form, the request is rejected. If *form-name* has not been defined for the system or if the user is not allowed to use the form, the request is rejected [see *lpforms*(ADM)]. When the **-d any** option is given, the request is printed on any printer that has the requested form mounted and can handle all other needs of the print request.

-H *special-handling*
Prints the request according to the value of *special-handling*. Acceptable values for *special-handling* are **hold**, **resume**, and **immediate**, as defined below:

hold Won't print the request until notified. If already printing, stops it. Other print requests will go ahead of a held request until it is resumed.

resume Resumes a held request. If it had been printing when held, it will be the next request printed, unless subsequently bumped by an **immediate** request.

immediate
(Available only to LP administrators)
Prints the request next. If more than one request is assigned **immediate**, the requests are printed in the reverse order queued. If a request is currently printing on the desired printer, you have to put it on hold to allow the immediate request to print.

-m Sends mail [see *mail*(C)] after the files have been printed. By default, no mail is sent upon normal completion of the print request.

-n *number*

Prints *number* copies of the output (default is 1).

-o *option* Specifies printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **-o** keyletter more than once. The standard interface recognizes the following options:

nobanner

Does not print a banner page with this request. (The administrator can disallow this option at any time.)

nofilebreak

Does not insert a form feed between the files given if submitting a job to print more than one file.

length=*scaled-decimal-number*

Prints the output of this request with pages *scaled-decimal-number* lines long. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter "i" (for inches) or the letter "c" (for centimeters). For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with "i"). For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

This option cannot be used with the **-f** option.

width=*scaled-decimal-number*

Prints the output of this request with page-width set to *scaled-decimal-number* columns wide. (See the explanation above for *scaled-decimal-numbers*.) This option cannot be used with the **-f** option.

lpi=*scaled-decimal-number*

Prints this request for "lines per inch" with the line pitch set to *scaled-decimal-number* lines per inch. This option cannot be used with the **-f** option.

cpi=*scaled-decimal-number*

Prints this request for "characters per inch" with the character pitch set to *scaled-decimal-number* characters per inch. Character pitch can also be set to **pica** (representing 10 columns per inch) or **elite** (representing 12 columns per inch), or it can be **compressed**, which is as many columns as a printer can handle. There is no standard number of columns per inch for all printers; see the *terminfo*(F) database for the default character pitch for your printer. The **cpi** option cannot be used in conjunction with the **-f** option.

stty=*stty-option-list*
Set the printer with a list of options valid for the **stty** command. Enclose the list with quotes if it contains blanks.

-P *page-list*
Prints the page(s) specified in *page-list*. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

The *page-list* may consist of range(s) of numbers, single page numbers, or a combination of both. The pages will be printed in ascending order.

-q *priority-level*
Assigns this request *priority-level* in the printing queue. The values of *priority-level* range from 0, the highest priority, to 39, the lowest priority. If a priority is not specified, the default for the print service is used, as assigned by the system administrator.

-s Suppresses messages from *lp*(C) such as "request id is ...".

-S *character-set* [**-d any**]

-S *print-wheel* [**-d any**]
Prints this request using the specified *character-set* or *print-wheel*. If a form has been specified that requires a *character-set* or *print-wheel* other than the one specified with the **-S** option, the request is rejected.

For printers that take print wheels: if the *print-wheel* specified is not one listed by the administrator as acceptable for the printer involved in this request, the request is rejected unless the print wheel is already mounted on the printer. For printers that use selectable or programmable character sets: if the *character-set* specified is not one defined in the terminfo database for the printer [see *terminfo*(F)] or is not an alias defined by the administrator, the request is rejected.

When the **-d any** option is used, the request is printed on any printer that has the print wheel mounted or any printer that can select the character set and can handle all other needs of the request.

-t *title* Prints *title* on the banner page of the output. The default is no title.

-T *content-type* **[-r]**
While the printer type information tells the print service what type of printer is being added, the content type information tells the print service what types of files can be printed. Prints the request on a printer that can support the specified *content-type*. If no printer accepts this type directly, a filter will be used to convert the content into an acceptable type. If the **-r** option is specified, a filter will not be used. If **-r** is specified but no printer accepts the *content-type* directly, the request is rejected. If the *content-type* is not acceptable to any printer, either directly or with a filter, the request is rejected.

-w Writes a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

-y *mode-list*
Prints this request according to the printing modes listed in *mode-list*. The allowed values for *mode-list* are locally defined. This option can be used only if there is a filter available to handle it; if there is no filter, the print request will be rejected.

Canceling a Print Request

The *cancel* command cancels printer requests that were made by the *lp*(C) shell command. The shell command line arguments may be either *request-ids* [as returned by *lp*(C)] or *printer* names [for a complete list, use *lpstat*(C)]. Specifying a *request-id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request that is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

Special Options

-R Removes file after sending it.

-L Local printing option. Sends print job to printer attached to the terminal.

The file **/etc/default/lpd** contains the setting of the variable BANNERS, whose value is the number of pages printed as a banner identifying each printout. This is normally set to either 1 or 2.

The variables LPR and PRINTER can each be set to 'spooler' or 'local'. These variables let you send files to the spool printer or the terminal's local printer, respectively. The file **/usr/bin/spool** contains the 'spooler' setting for both variables. The file **/usr/bin/local** contains the 'local' setting. The following are a few examples of variable usage:

lp *-option* spooler
LPR=local
LPR=spooler
spool lp *-option device file*

Notes

Printers for which requests are not being accepted will not be considered when the destination is **any**. (Use the **lpstat -a** command to see which printers are accepting requests.) On the other hand, if a request is destined for a class of printers and the class itself is accepting requests, *all* printers in the class will be considered, regardless of their acceptance status, as long as the printer class is accepting requests.

Warning

For printers that take mountable print wheels or font cartridges, if you do not specify a particular print wheel or font with the -S option, whichever happens to be mounted at the time your request prints will be used. Use the **lpstat -p -l** command to see what print wheels are available. For printers that have selectable character sets, you will get the standard set if you don't give the -S option.

Files

/usr/spool/lp/*
/etc/default/lpd

See Also

enable(C), lpstat(C), mail(C), accept(ADM), lpadmin(ADM), lpfilter(ADM), lpforms(ADM), lpsched(ADM), lpusers(ADM), terminfo(F)

Standards Conformance

cancel and *lp* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

lprint

print to a printer attached to the user's terminal

Syntax

lprint [-] [file]

Description

lprint(C) accepts a filename to print or - to read from the keyboard. If the terminal has local printing abilities, it will then print the file to a printer attached to the printer port of the terminal.

This command uses the file **/etc/termcap.**

Options

- Tells *lprint* to use the standard input for printing.

The variables LPR and PRINTER can each be set to 'spooler' or 'local'. These variables let you send files to the spool printer or the terminal's local printer, respectively. The file **/usr/bin/spool** contains the 'spooler' setting for both variables. The file **/usr/bin/spool** contains the 'local' setting. The following are a few examples of variable usage:

lp *-option* spooler
LPR=local
LPR=spooler
spool lp *-option device file*

Files

/etc/termcap
/usr/bin/spool
/usr/bin/local

Notes

Only certain terminals have entries in /etc/termcap with this capability already defined (for example, Tandy's DT-100 and DT-1, and Hewlett-Packard's HP-92).

To add attached printer capability to the termcap file for a different terminal, add entries for PN (start printing) and PS (end printing) with the appropriate control or escape characters for your terminal.

Terminal communications parameters (such as baud rate and parity) must be set up on the terminal by the user.

See Also

"Using Printers" in the *System Administrator's Guide*

Value Added

lprint is an extension of AT&T System V provided by Altos UNIX System V.

lpstat

print information about status of LP print service

Syntax

lpstat [options]

Description

lpstat prints information about the current status of the LP print service.

If no options are given, then *lpstat* prints the status of all requests made to *lp*(C) by the users. Any arguments that are not options are assumed to be *request-ids* (as returned by *lp*), printers, or printer classes. *lpstat* prints the status of such requests, printers, or printer classes. Options may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

-u "user1, user2, user3"

Specifying "all" after any keyletters that take "list" as an argument causes all information relevant to the keyletter to be printed. For example, the command

lpstat -o all

prints the status of all output requests.

-a [*list*] Print acceptance status (with respect to *lp*) of destinations for requests [see *accept*(ADM)]. *list* is a list of intermixed printer names and class names; the default is **all**.

-c [*list*] Print class names and their members. *list* is a list of class names; the default is **all**.

-d Print the system default destination for *lp*.

-f [*list*] [-l]
Print a verification that the forms in *form-list* are recognized by the LP print service. The -l option will list the form descriptions.

-o [*list*] [**-l**]
: Print the status of output requests. *list* is a list of intermixed printer names, class names, and *request-ids*; the default is **all**. The **-l** option gives a more detailed status of the request.

-p [*list*] [**-D**] [**-l**]
: Print the status of printers named in *list*. If the **-D** option is given, a brief description is printed for each printer in *list*. If the **-l** option is given, a full description of each printer's configuration is given, including the form mounted, the acceptable content and printer types, a printer description, the interface used, and so on.

-r Print the status of the LP request scheduler.

-s Print a status summary, including the system default destination, a list of class names and their members, a list of printers and their associated devices, a list of all forms currently mounted, and a list of all recognized character sets and print wheels.

-S [*list*] [**-l**]
: Print a verification that the character sets or the print wheels specified in *list* are recognized by the LP print service. Items in *list* can be character sets or print wheels; the default for the list is **all**. If the **-l** option is given, each line is appended by a list of printers that can handle the print wheel or character set. The list also shows whether the print wheel or character set is mounted or specifies the built-in character set into which it maps.

-t Print all status information.

-u [*list*] Print status of output requests for users. *List* is a list of login names. The default is **all**.

-v [*list*] Print the names of printers and the path names of the devices associated with them. *list* is a list of printer names. The default is **all**.

Files

/usr/spool/lp/*

See Also

enable(C), lp(C)

Standards Conformance

lpstat is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ls

gives information about contents of directories

Syntax

ls [**-ACFRabcdfgilmnopqrstux**] [names]

Description

For each directory named, *ls* lists the contents of that directory; for each file named, *ls* repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, file arguments are processed before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **-C** and **-x** options enable multi-column formats, and the **-m** option enables stream output format in which files are listed across the page, separated by commas. In order to determine output format for the **-C**, **-x**, and **-m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *termcap* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

There are many options:

-A List all entries. Entries whose name begin with a period (.) are listed. Does not list current directory (.) and directory above (..).

-a Lists all entries. Entries whose name begin with a period (.) are listed.

-R Recursively lists subdirectories encountered.

-d If an argument is a directory, lists only its name (not its contents); often used with **-l** to get the status of a directory.

-C Multi-column output with entries sorted down the columns.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format.

-l Lists in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will contain the major and minor device numbers, rather than a size.

-n The same as **-l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

-o The same as **-l**, except that the group is not printed.

-g The same as **-l**, except that the owner is not printed.

-r Reverses the order of sort to get reverse alphabetic or oldest first, as appropriate.

-t Sorts by time modified (latest first) instead of by name.

-u Uses time of last access instead of last modification for sorting use with the **-t** option.

-c Uses time of last modification of the inode (file created, mode changed, etc.) for sorting use with **-t** option.

-p Puts a slash (/) after each filename if that file is a directory.

-F Puts a slash (/) after each filename if that file is a directory and puts an asterisk (*) after each filename if that file is executable.

-b Forces printing of non-graphic characters to be in the octal \ddd notation.

-q Forces printing of non-graphic characters in file names as the character (?).

-i For each file, prints the inode number in the first column of the report.

-s Gives size in blocks, including indirect blocks, for each entry.

-f Forces each argument to be interpreted as a directory and lists the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**. The order is the order in which entries appear in the directory.

The mode printed under the **-l** option consists of 11 characters. The first character is:

- If the entry is an ordinary file.

d If the entry is a directory.

b If the entry is a block special file.

c If the entry is a character special file.

p If the entry is a named pipe.

s If the entry is a semaphore.

m If the entry is a shared data (memory) file.

The next 9 characters are interpreted as 3 sets of 3 bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the 3 characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r If the file is readable.

w If the file is writable.

x If the file is executable.

- If the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **-**) is **t** if the 1000 (octal) bit of the mode is on. See *chmod*(C) for the meaning of this mode. The indications for set-ID and the 1000 bit of the mode are capitalized if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks including indirect blocks is printed.

Files

/etc/passwd	Gets user IDs for ls -l and ls -o
/etc/group	Gets group IDs for ls -l and ls -g
/etc/termcap	Gets terminal information

See Also

chmod(C), coltbl(M), find(C), l(C), lc(C), locale(M), termcap(F)

Notes

Sorts according to the collating sequenced defined by the locale.

Newline and tab are considered printing characters in filenames.

Unprintable characters in filenames may confuse the columnar output options.

This utility reports sizes in 512 byte blocks. *ls -s* interprets 1 block from a 1024 byte block system as 2 of its own 512 byte blocks. Thus a 500 byte file is interpreted as 2 blocks rather than 1.

Standards Conformance

ls is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

machid: i386

get processor type truth value

Syntax

i386

Description

If the machine is a 386, the *i386* command will return a true value (exit code of 0), otherwise it will return a false (non-zero) value. These type of commands are often used within makefiles [see *make*(CP)] and shell procedures [see *sh*(C)] to increase portability.

See Also

sh(C), test(C), true(C), make(CP)

mail

interactive message processing system

Syntax

mail [*options*] [*name...*]

Description

mail provides a comfortable, flexible environment for sending and receiving messages electronically. For reading mail, *mail* provides commands to facilitate saving, deleting, and responding to messages. For sending mail, *mail* allows editing, reviewing, and other modification of the message as it is entered.

Many of the remote features of *mail* will only work if the UUCP package is installed on your system.

Incoming mail is stored in a standard file for each user, called the **mailbox** for that user. When *mail* is called to read messages, the **mailbox** is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the **mbox** and is normally located in the user's HOME directory (see **MBOX** under Environment Variables). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **-f** option of the *mail* command. Messages in the secondary file can then be read or otherwise processed using the same commands as in the primary **mailbox**. This gives rise to the notion of a current **mailbox**.

On the command line, *options* start with a dash (-) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mail* attempts to read messages from the **mailbox**. Command-line options are:

-e Test for presence of mail. *mail* prints nothing and exits with a successful return code if there is mail to read.

-f [*filename*] Read messages from *filename* instead of **mailbox**. If no *filename* is specified, the **mbox** is used.

-F Record the message in a file named after the first recipient. Overrides the **record** variable, if set (see Environment Variables).

-h *number* The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. (See **addsopt** under Environment Variables).

-H Print header summary only.

-i Ignore interrupts. (See **ignore** under Environment Variables).

-n Do not initialize from the system default **mailrc** file.

-N Do not print initial header summary.

-r *address* Pass *address* to network delivery software. All tilde commands are disabled. (See **addsopt** under Environment Variables).

-s *subject* Set the Subject header field to *subject*.

-u *user* Read *user*'s **mailbox**. This is only effective if *user*'s **mailbox** is not read protected.

-U Convert *uucp* style addresses to internet standards. Overrides the **conv** environment variable. (See **addsopt** under Environment Variables).

When reading mail, *mail* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mail* can accept regular commands (see Commands below). When sending mail, *mail* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A subject longer than 1024 characters will cause *mail* to dump core.) As the message is typed, *mail* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See Tilde Escapes for a summary of these commands.

At any time, the behavior of *mail* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the set and unset commands. See Environment Variables below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to be undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(C), for recording outgoing mail on paper. Alias groups are set by the **alias** command (see Commands below) and are lists of recipients of any type.

Regular commands are of the form:

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the tilde escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

n	Message number **n**.
.	The current message.
^	The first undeleted message.
$	The last message.
*****	All messages.
n-m	An inclusive range of message numbers.
user	All messages from *user*.
/string	All messages with *string* in the subject line (case ignored).
:*c*	All messages of type *c*, where *c* is one of:

d deleted messages
n new messages
o old messages
r read messages
u unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions [see *sh*(C)]. Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mail* tries to execute commands from the optional system-wide file (**/usr/lib/mail/mailrc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **Copy**, **edit**, **forward**, **Forward**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, and **visual**. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional and must be constructed locally.

Commands

The following is a complete list of *mail* commands:

!*shell-command*
Execute shell command and return. (See **SHELL** under Environment Variables).

comment
Null command (comment). This may be useful in **.mailrc** files.

=
Print the current message number.

?
Print a summary of commands.

alias *alias name* ...
group *alias name* ...
Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the **.mailrc** file.

alternates *name* ...
Declare a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, alternates prints the current list of alternate names. (See **allnet** under Environment Variables).

cd [*directory*]
chdir [*directory*]
Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*
Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.

Copy [*msglist*]
Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.

delete [*msglist*]

Delete messages from the **mailbox**. If **autoprint** is set, the next message after the last one deleted is printed (see Environment Variables).

discard [*header-field* ...]
ignore [*header-field* ...]

Suppress printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc". The fields are included when the message is saved. The Print and Type commands override these commands.

dp [*msglist*]
dt [*msglist*]

Delete the specified messages from the **mailbox** and print the next message after the last one deleted. Roughly equivalent to a **delete** command followed by a **print** command.

echo *string* ...

Echo the given strings [like *echo*(C)].

edit [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the EDITOR variable is used to get the name of the editor (see Environment Variables). Default editor is *ed*(C).

exit
xit

Exit from *mail* without changing the **mailbox**. No messages are saved in the **mbox** (see also **quit**).

file [*filename*]
folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current **mailbox**.
%**user**	the **mailbox** for **user**.
#	the previous file.
&	the current **mbox**.

Default file is the current **mailbox**.

folders

Print the names of the files in the directory set by the **folder** variable (see Environment Variables).

forward [*message*] *name* ...

Forward the specified message to the specified users, shifting the forwarded text to the right one tab stop.

Forward [*message*] *name* ...
Forward the specified message to the specified users, with no indentation.

from [*msglist*]
Prints the header summary for the specified messages.

group *alias name* ...
See alias.

headers [+ | - | *msglist*]
Lists the current range of headers. The **screen** variable sets the number of headers per page (see Environment Variables). If a "+" argument is given, then the next page is printed, and if a "-" argument is given, the previous page is printed. Both "+" and "-" can take a number to view a particular window. If a message list is given, it prints the specified headers, disregarding all windowing. See also the **z** command.

help
Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
Holds the specified messages in the **mailbox**.

if *s* | *r*
mail-commands
else
mail-commands
endif
Conditional execution, where *s* causes the first mail commands, up to an else or endif to be executed if the program is in *send* mode, and *r* causes the mail commands to be executed only in *receive* mode. The *mail-commands* after the else are executed if the program is in the opposite mode from the one indicated. Useful in the **.mailrc** file.

ignore *header-field* ...
discard *header-field* ...
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc". All fields are included when the message is saved. The Print and Type commands override this command.

list
Prints all commands available. No explanation is given.

lpr [*msglist*]
Print the specified messages on the lineprinter.

mail *name ...*
Mail a message to the specified users.

Mail *name*
Mail a message to the specified user and record a copy of it in a file named after that user.

mbox [*msglist*]
Arrange for the given messages to end up in the standard **mbox** save file when *mail* terminates normally. See the **exit** and **quit** commands.

next [*message*]
Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]
Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the **cmd** variable. If the **page** variable is set, a form feed character is inserted after each message (see Environment Variables).

preserve [*msglist*]
See hold.

Print [*msglist*]
Type [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** command.

print [*msglist*]
type [*msglist*]
Print the specified messages. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable are paged through the command specified by the **PAGER** variable. The default command is *more*(C) (see Environment Variables).

quit
Exit from *mail*, storing messages that were read in **mbox** and unread messages in the **mailbox**. Messages that have been explicitly saved in a file are deleted from the mailbox.

Reply [*msglist*]
Respond [*msglist*]
Reply to the specified message, including all other recipients of the message. If **record** is set to a file name, the response is saved at

the end of that file (see Environment Variables).

reply [*message*]
respond [*message*]
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If **record** is set to a file name, the response is saved at the end of that file (see Environment Variables).

Save [*msglist*]
Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the Copy commands and **outfolder** (Environment Variables).

save [*filename*]
save [*msglist*] *filename*
Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the **mailbox** when *mail* terminates unless **keepsave** is set (see also Environment Variables and the **exit** and **quit** commands).

set
set *name*
set *name=string*
set *name=number*
Define a variable called *name*. The variable may be given a null, string, or numeric value. **Set** by itself prints all defined variables and their values. See Environment Variables for detailed descriptions of the *mail* variables.

shell
Invoke an interactive shell (see **SHELL** under Environment Variables).

size [*msglist*]
Print the size in characters of the specified messages.

source *filename*
Read commands from the given file and return to command mode.

top [*msglist*]
Print the top few lines of the specified messages. If the **toplines** variable is set, it is taken as the number of lines to print (see Environment Variables). The default is 5.

touch [*msglist*]
Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the **mbox**, or the file specified in the **MBOX** environment variable, upon normal termination. See **exit** and **quit**.

Type [*msglist*]
See Print.

type [*msglist*]
See print.

undelete [*msglist*]
Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If **autoprint** is set, the last message of those restored is printed (see Environment Variables).

unset *name* ...
Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable), then it cannot be erased.

version
Prints the current version and release date.

visual [*msglist*]
Edit the given messages with a screen editor. The messages are placed in a temporary file and the **VISUAL** variable is used to get the name of the editor (see Environment Variables).

write [*msglist*] *filename*
Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the save command.

xit
See exit. quit).

z[+ | -]
Scroll the header display forward or backward one full screen. The number of headers displayed is set by the **screen** variable (see Environment Variables).

Tilde Escapes

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See **escape** under Environment Variables for changing this special character.

~! *shell-command*
Execute the shell command and return.

~.
Simulate end of file (terminate message input).

~: *command*
~_ *command*
Perform the command-level request. Valid only when sending a message while reading mail.

~?
Print a summary of tilde escapes.

~A
Insert the autograph string **Sign** into the message (see Environment Variables).

~a
Insert the autograph string **sign** into the message (see Environment Variables).

~b *name ...*
Add the *names* to the blind carbon copy (Bcc) list.

~c *name ...*
Add the *names* to the carbon copy (Cc) list.

~d
Read in the *dead.letter* file. (See **DEAD** under Environment Variables for a description of this file.)

~e
Invoke the editor on the partial message. (See **EDITOR** under Environment Variables.)

~f [*msglist*]
Forward the specified messages. The messages are inserted into the message without alteration.

~h
Prompt for Subject line and To, Cc, Bcc, and Return-Receipt-to lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *variable*
Insert the value of the named variable into the text of the message. For example, ~A is equivalent to '~i Sign.' Environment variables set and exported in the shell are also accessible by ~i.

~M [*msglist*]
Insert the specified messages into the letter, with no indentation. Valid only when sending a message while reading mail.

~m [*msglist*]
Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p
Print the message being entered.

~q
Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. (See **DEAD** under Environment Variables).

~r *filename*
~~< *filename*
~~< !*shell-command*
Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string* ...
Set the subject line to *string*.

~t *name* ...
Add the given *names* to the To list.

~v
Invoke a preferred screen editor on the partial message. (See also **VISUAL** under Environment Variables.)

~w *filename*
Write the partial message onto the given file, without the header.

~x
Exit as with **~q** except the message is not saved in *dead.letter*.

~| *shell-command*
Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

Environment Variables

The following are environment variables taken from the execution environment and are not alterable within *mail*.

HOME=*directory*
The user's base of operations.

MAILRC=*filename*
The name of the start-up file. Default is **$HOME/.mailrc.**

The following variables are internal *mail* variables. They may be imported from the execution environment or set via the **set** command at any time. The **unset** command may be used to erase variables.

addsopt
Enabled by default. If **/bin/mail** is not being used as the deliverer, **noaddsopt** should be specified. (See Notes below)

allnet
All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet.** See also the **alternates** command and the **metoo** variable.

append
Upon termination, append messages to the end of the **mbox** file instead of prepending them. Default is **noappend.**

askcc
Prompt for the Cc list after message is entered. Default is **noaskcc.**

asksub
Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint
Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint.**

bang
Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(C). Default is **nobang.**

chron
Causes messages to be displayed in chronological order. The default is reverse chronological order (most recent message first). See also **mchron** below.

cmd=*shell-command*
Set the default command for the **pipe** command. Not set by default.

conv=*conversion*
Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also the **sendmail** variable and the **-U** command-line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the value of the **PAGER** variable (*more*(C) by default). Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt. Default is $HOME/dead.letter.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug.**

dot
Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot.**

EDITOR=*shell-command*
The command to run when the edit or **~e** command is used. Default is *ed*(C).

escape=*c*
Substitute *c* for the ~ escape character. Takes effect with next message sent.

folder=*directory*
The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mail* command line, **folder** must be an exported *sh* environment variable. There is no default for the **folder** variable. See also **outfolder** below.

header
Enable printing of the header summary when entering *mail.* Enabled by default.

hold
Preserve all messages that are read in the **mailbox** instead of putting them in the standard **mbox** save file. Default is **nohold.**

ignore
Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore.**

ignoreeof
Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the **~.** command. Default is **noignoreeof.** See also the **dot** variable above.

keep
When the **mailbox** is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave
Keep messages that have been saved in other files in the **mailbox** instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*
The name of the file to save messages which have been read. The xit command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

mchron
Causes message headers to be listed in numerical order (most recently received first), but displayed in chronological order. See also **chron** above.

metoo
If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*
The command (and options) to use when listing the contents of the **folder** directory. The default is *ls*(C).

onehop
When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
Record outgoing messages in files located in the directory specified by the **folder** variable unless the path name is absolute. Default is **nooutfolder**. See the **folder** variable above and the Save and Copy commands.

page
Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
Use *shell-command* as a filter for paginating output. This can also be used to specify the options to be used. Default is *more*(C).

prompt=*string*
Set the *command mode* prompt to *string*. Default is **?** .

quiet
Refrain from printing the opening message and version when entering *mail.* Default is **noquiet.**

record=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also **outfolder** above.

save
Enable saving of messages in *dead.letter* on interrupt or delivery error. See **DEAD** for a description of this file. Enabled by default.

screen=*number*
Sets the number of lines in a full screen of headers for the **headers** command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is */bin/rmail*(C).

sendwait
Wait for background mailer to finish before returning. Default is **nosendwait.**

SHELL=*shell-command*
The name of a preferred command interpreter. Default is *sh*(C).

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
The variable inserted into the text of a message when the **~a** (autograph) command is given. Not set by default (see **~i** under Tilde Escapes).

Sign=*string*
The variable inserted into the text of a message when the **~A** command is given. Not set by default (see also **~i** under Tilde Escapes).

toplines=*number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. Default is *vi*(C).

Files

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/spool/mail	post office directory
/usr/lib/mail/mail.help*	help message files
/usr/lib/mail/mailrc	optional global start-up file
/tmp/R[emqsx]*	temporary files

See Also

ls(C), mail(C), more(C)

Notes

The **-h**, **-r** and **-U** options can be used only if *mail* is built with a delivery program other than */bin/mail*.

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be unset.

The full internet addressing is not fully supported by *mail*. The new standards need some time to settle down.

A line consisting only of a "." is treated as the end of the message.

Standards Conformance

mail is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

man

prints reference pages in this guide

Syntax

man [**-afbcw**] [**-t***proc*] [**-p***pager*] [**-d***dir*] [**-T***term*] [*section*] [*title*]

/usr/lib/manprog file

Description

The *man* program locates and prints the named *title* from the designated reference *section*. For historical reasons, ''page'' is often used as a synonym for ''entry'' in this context.

Since Altos UNIX System V commands are given in lowercase, the *title* is always entered in lowercase. If no *section* is specified, the whole guide is searched for *title* and the first occurrence of it is printed. You can search for a group of *sections* by separating the section names with colons (:) on the command line.

The options and their meanings are:

-a ''All'' mode. Displays all matching titles. Incompatible with the **-f** option.

-f ''First'' mode. Displays only the first matching title. Incompatible with **-a** option. This is the default mode for **man**(C).

-b Leaves blank lines in output. **nroff** pads entries with blank lines for line printer purposes. **man** normally filters out these excess blank lines. Normally, **man** does not display more than 2 consecutive blank lines. The **-b** flag leaves blank lines in the CRT output.

-c Causes **man** to invoke **col**(C). Note that **col** is invoked automatically by **man** unless *term* is one of the following: **300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620**, and **X**.

-w Prints on the standard output only the *pathnames* of the entries.

-t*proc*	Indicates that if an unprocessed manual page is available, it is to be passed to *proc* for formatting. *proc* can be any command script in */usr/man/bin* or an absolute filename of a text processing program elsewhere on the system, for example */bin/nroff*. The scripts in */usr/man/bin* invoke the actual processing programs with the correct flags and arguments. The default processor is */usr/man/bin/nr*, which invokes */bin/nroff* and produces output that safely prints on any terminal. The text is also preprocessed by **eqn** and **tbl** as a default. Note that the operating system does not include these formatting programs; you must install them yourself or specify alternatives with the **-t** option.
-p*pager*	Selects paging program *pager* to display the entry. Paging systems such as **more**(C), **pg**(C), **cat**(C), or any custom pagers that you may have are valid arguments for this flag. The default pager, **pg**(C), is set in */etc/default/man*.
-d*dir*	Specifies directory *dir* to be added to the search path for entries. You can specify several directories to be searched by separating the directory names with colons (:) on the command line.
-T*term*	Format the entry and pass the given *term* value to the processing program, then print it on the standard output (usually, the terminal), where *term* is the terminal type (see **term**(M) and the explanation below).

Section Names

The names and general descriptions of the available manual sections are:

ADM	System Administration
C	Commands
M	Miscellaneous
F	File Formats
HW	Hardware Dependent
S	Subroutines and Libraries
CP	Programming Commands
DOS	DOS Subroutines and Libraries
K	Kernel Routines
NSL	Network Services Library
STR	STREAMS
XNX	XENIX Cross-development

LOCAL Local utilities for your system

You can add other section names as you desire. Each new section, however, must follow the standard section directory structure. The LOCAL directory is shipped without contents, as no LOCAL manual pages are included.

/usr/man Directory Structure

The source files for the **man**(C) program are kept in the directory */usr/man*. Each **man** section is comprised of two directories, and there is a directory called *bin* for programs and shell scripts related to **man**(C). There is also an index file called *index* in */usr/man*. This index is a list of all Altos UNIX System V commands and their sections.

Each manual section has two directories in */usr/man*. These directories are called *man* and *cat*, plus the name of the section as a suffix. For example, the **C** manual section is comprised of two directories, *man.C* and *cat.C*, both located in */usr/man*.

The unprocessed source text is in the *man* directory and the printable processed output is in the *cat* directory. When a title is requested, both directories are checked. The most recent copy of the manual page is used as the current copy. If the most recent title is in the source text directory and it is processed by the default processor with the default terminal type, a display copy of the output is placed in the *cat* directory for future use. Note that a file that must be processed takes longer to appear on the screen than a display copy.

Environment Variables

There is a shell environment variable for use with the **man**(C) utility. This variable is called **MANPATH** and it is used to change or augment the path **man**(C) searches for entries. Multiple directories set with this variable must be delimited by colon characters (:). If the **MANPATH** environment variable is present, the directories are searched in the order that they appear. */usr/man* must appear in the **MANPATH** list to be included. If you set this environment variable, it supersedes the **MANPATH** entry in the */etc/default/man* file. Alternate subdirectories are expected to have the same form as the default directories in */usr/man*.

/etc/default/man

There is a file called *man* in the */etc/default* directory that contains the default settings for the **man** utility. The following options are set in */etc/default/man*:

```
PAGER=/usr/bin/pg
MANPATH=/usr/man
TERM=lp
ORDER=ADM:C:S:CP:M:F:HW:DOS:LOCAL
MODE=FIRST
PROC=nr
```

You can select a different paging system, search path, terminal type, search order, mode, and processor for the **man**(C) system by changing the information in this file.

To change the search order for manual sections, edit the list following the **ORDER** variable. Be certain the section names are separated with colons (:). Section names not present in **ORDER** are searched in arbitrary order after those specified in */etc/default/man.*

Creating New Manual Entries

You can create new manual pages for utilities and scripts that you have developed. Use an existing manual page as an example of manual page structure. Use the **man** macros to format your manual page. Note that the operating system does not include **nroff** and the related family of formatting utilities. You must install them separately or specify another formatter with the **-t** option.

You must be logged in as root (the "Super-User") to place a new manual page in your */usr/man* directory structure. Place your new page in */usr/man/man.LOCAL* while logged in as root and view it using the **man**(C) command, since only root has write permission for the cat-able directories. Once **man** has produced the cat-able output, any user can view the new page in the same manner as any other on line manual page.

Additionally, you can create your own custom sections by creating another manual directory and putting it in the **MANPATH**. For example, if subdirectories *man.X* and *cat.X* are present, then **man**(C) recognizes that **X** is a valid manual section.

If you wish to use another text processing program (such as **troff** to process your custom manual pages, use the **-t***proc* flag of **man**. *proc* can be any shell script in */usr/man/bin.* To place a cat-able copy of the manual page in the *cat* directory, use the **tee**(C) command to send the output to a file, as well as to the standard output. Your command should have the form:

man -t*proc filename* | **tee** *pathname*

In the above example, *proc* is the text processing script, *filename* is the manual page source file, and *pathname* is the path of the directory for the cat-able output.

Custom manual sections can have an index, if the format is the same as the index in */usr/man*. **man**(C) uses the index to locate multiple commands that are listed on the same page as well as commands that have pages in several different sections.

The man Macro Package

The **man** macro package is located in */usr/lib/macros/an*. It is included for use with the **nroff/troff** formatting package, which must be separately installed. There are 15 basic macros in the package. Here is a table of the macros and brief descriptions of their functions:

Macro	Description
.TH *title*	Title Heading
.SH *title*	Section Heading
.SS *title*	Subsection Heading
.SM *text*	Reduce Point Size
.PP	New Paragraph
.IP	Indented Paragraph
.HP	Hanging Paragraph
.TP	Tagged Paragraph
.RS *n*	Relative Indent
.RE	Release Relative Indent
.I *text*	Italic Font
.B *text*	Bold Font
.R *text*	Roman Font
.PM	Proprietary Mark (copyright)

See Also

environ(M), term(F)

Notes

All entries are supposed to be reproducible either on a typesetter or on a terminal. However, on a terminal some information, such as **eqn** and **tbl** output, is either lost or approximated as it cannot be exactly reproduced.

mesg

permits or denies messages sent to a terminal

Syntax

mesg [**n**] [**y**]

Description

mesg with argument **n** forbids messages via *write*(C) by revoking nonuser write permission on the user's terminal. *mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

Files

/dev/tty*

See Also

write(C)

Diagnostics

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

Standards Conformance

mesg is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

mkdir

makes a directory

Syntax

mkdir [**-m** mode] [**-p**] dirname ...

Description

The *mkdir* command creates the named directories in mode 777 [possibly altered by *umask*(C)].

Standard entries in a directory (e.g., the files **.**, for the directory itself, and **..**, for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

-m This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(C).

-p With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

See Also

sh(C), rm(C), rmdir(C), umask(C), mkdir(S)

Diagnostics

The *mkdir* command returns exit code 0 if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero. An error code is stored in *errno* .

Standards Conformance

mkdir is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

mknod

builds special files

Syntax

/etc/mknod name [**c** | **b**] major minor

/etc/mknod name **p**

/etc/mknod name **s**

/etc/mknod name **m**

Description

mknod makes a directory entry and corresponding inode for a special file. The first argument is the *name* of the entry. In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g., unit, drive, or line number), which may be either decimal or octal.

The assignment of major device numbers is specific to each system. Major device numbers can be found in the system source file **/etc/conf/cf.d/mdevice**.

mknod can also be used to create named pipes with the **p** option, semaphores with the **s** option, and shared data (memory) with the **m** option.

Only the super-user can use the first form of the syntax.

See Also

mknod(S)

Standards Conformance

mknod is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

mnt, umnt

mount a filesystem

Syntax

/usr/bin/mnt [-urat] [directory]

/usr/bin/umnt *directory*

Description

mnt allows users other than the super-user to access the functionality of the *mount*(ADM) command to mount selected filesystems. The super-user can define how and when a filesystem mount is permitted via special entries in the */etc/default/filesys* file.

The filesystem requirements are the same as defined for *mount*(ADM).

umnt removes the mountable filesystem previously mounted in *directory* .

mnt is invoked from the */etc/rc* scripts with the **-r** and possibly the **-a** flag to mount filesystems when the system comes up multiuser. The **-a** flag is used when the system has autobooted. Neither of these flags should be specified during normal use.

The **-t** flag displays the contents of /etc/default/filesys.

The **-u** flag forces *mnt* to behave like *umnt.*

Options

The following options can be defined in the */etc/default/filesys* entry for a filesystem:

bdev=/dev/*device*	Name of block device associated with the filesystem.
cdev=/dev/*device*	Name of character (raw) device associated with the filesystem.
mountdir=/*directory*	The directory the filesystem is to be mounted on.

desc=*name* — A string describing the filesystem.

passwd=*string* — An optional password prompted for at mount request time. Cannot be a simple string; must be in the format of /etc/passwd. (See **Notes**.)

fsck=**yes, no, dirty, prompt**
If yes/no, tells explicitly whether or not to run *fsck*. If dirty, *fsck* is run only if the filesystem requires cleaning. If prompt, the user is prompted for a choice. If no entry is given, the default value is dirty.

fsckflags=*flags* — Any flags to be passed to *fsck*.

rcfsck=**yes, no, dirty, prompt**
Similar to fsck entry, but only applies when the **-r** flag is passed.

maxcleans=*n* — The number of times to repeat cleaning of a dirty filesystem before giving up. If undefined, default is 4.

mount=**yes, no, prompt**
If yes or no, users are allowed or disallowed to mount the filesystem, respectively. If prompt, the user specifies whether the filesystem should be mounted.

rcmount=**yes, no, prompt**
If yes, the filesystem is mounted by */etc/rc2* when the system comes up multiuser. If no, the filesystem is never mounted by */etc/rc2*. With prompt, a query is displayed at boot time to mount the filesystem.

mountflags=*flags* — Any flags to be passed to *mount*.

prep=**yes, no, prompt** — Indicates whether any prepcmd entry should always be executed, never executed, or executed as specified by user.

prepcmd=*command* — An arbitrary shell command to be invoked immediately following password check and prior to running *fsck*.

init=**yes, no, prompt** — Indicates whether an initcmd entry should always be executed, never be executed, or executed as specified by user.

initcmd=*command*	An optional, arbitrary shell command to be invoked immediately following a successful mount.

Any entries containing spaces, tabs, or newlines must be contained in double quotes (").

The only mandatory entries in */etc/default/filesys* are **bdev** and **mountdir**. The **prepcmd** and **initcmd** options can be used to execute another command before or after mounting the filesystem. For example, **initcmd** could be defined to send mail to root whenever a given filesystem is mounted.

When invoked without arguments, *mnt* attempts to mount all filesystems that have the entries **mount=yes** or **mount=prompt**.

Examples

The following is a sample */etc/default/filesys* file:

```
bdev=/dev/root  cdev=/dev/rroot  mountdir=/  \
desc="The Root Filesystem"  rcmount=no  mount=no

bdev=/dev/u  cdev=/dev/ru  mountdir=/u  rcmount=yes  \
fsckflags=-y  desc="The User Filesystem"

bdev=/dev/x  cdev=/dev/rx  mountdir=/u  rcmount=no  \
mount=yes  fsckflags=-y  desc="The Extra Filesystem"
```

Of the examples above, only */x* is mountable by the user.

Files

/etc/default/filesys	Filesystem data

See Also

mount(ADM), default(F)

Diagnostics

mnt will fail if the filesystem to be mounted is currently mounted under another name.

Busy filesystems cannot be unmounted with *umnt*. A filesystem is busy if it contains an open file or if a user's present working directory resides within the filesystem.

Notes

Some degree of validation is done on the filesystem, however it is generally unwise to mount corrupt filesystems.

In order to create a password for a filesystem, the system administrator must run the *passwd*(C) command using the **-f** option.

Value Added

mnt is an extension of AT&T System V provided by Altos UNIX System V.

more

views a file one screen full at a time

Syntax

more [**-cdflsuvw**] [-*n*] [+linenumber] [+/pattern] [name ...]

Description

This filter allows examination of a continuous text one screen full at a time. It normally pauses after each full screen, displaying:

--More--

at the bottom of the screen. If the user then presses a carriage return, one more line is displayed. If the user presses the SPACE bar, another full screen is displayed. Other possibilities are described below.

The command line options are:

-*n* An integer which is the size (in lines) of the window which *more* will use instead of the default.

-c *more* draws each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

-d *more* prompts with the message ''Hit space to continue, Rubout to abort" at the end of each full screen. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be inexperienced.

-f This option causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul,* since the latter may generate escape sequences. These escape sequences contain characters that would ordinarily occupy screen positions, but do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are and fold lines erroneously.

-l Does not treat Ctrl-L (form feed) specially. If this option is not given, *more* pauses after any line that contains a Ctrl-L, as if the end of a full screen has been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

-s Squeezes multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

-u Normally, *more* handles underlining, such as that produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The **-u** option suppresses this processing.

-v Normally, *more* ignores control characters that it does not interpret in some way. The *-v* option causes these to be displayed as *^C* where *C* is the corresponding printable ASCII character. Non-printing non-ASCII characters (with the high bit set) are displayed in the format M-*C*, where *C* is the corresponding character without the high bit set. If output is not going to a terminal, *more* does not interpret control characters.

-w Normally, *more* exits when it comes to the end of its input. With -*w* however, *more* prompts and waits for any key to be struck before exiting.

+*linenumber*
Starts up at *linenumber*.

+*/pattern*
Starts up two lines before the line containing the regular expression *pattern*.

more looks in the file **/etc/termcap** to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

more looks in the environment variable *MORE* to preset any flags desired. For example, if you prefer to view files using the **-c** mode of operation, the shell command "MORE=-c" in the **.profile** file causes all invocations of *more* to use this mode.

If *more* is reading from a file, rather than a pipe, a percentage is displayed along with the "--More--" prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be entered when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1 where not specified otherwise):

i <space>
Displays *i* more lines, (or another full screen if no argument is given).

Ctrl-D
Displays 11 more lines (a ''scroll''). If *i* is given, then the scroll size is set to *i*.

d Same as Ctrl-D.

i z Same as entering a space except that *i*, if present, becomes the new window size.

i s Skips *i* lines and displays a full screen of lines.

i f Skips *i* full screens and displays a full screen of lines.

q or Q
Exits from *more*.

= Displays the current line number.

v Starts up the screen editor *vi* at the current line. Note that *vi* may not be available with your system.

h or ?
Help command; Gives a description of all the *more* commands.

i /expr
Searches for the *i* th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a full screen is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

i n Searches for the *i* th occurrence of the last regular expression entered.

' (Single quotation mark) Goes to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*command*
Invokes a shell with *command*. The characters % and ! in ''command" are replaced with the current filename and the previous shell command respectively. If there is no current filename, % is not expanded. The sequences ''\%'' and ''\!'' are replaced by ''%'' and ''!'' respectively.

i :n
Skips to the *i* th next file given in the command line (skips to last file if *i* doesn't make sense).

i :p
Skips to the *i* th previous file given in the command line. If this command is given in the middle of printing out a file, *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell rings and nothing else happens.

:f Displays the current filename and line number.

:q or :Q
Exits from *more* (same as q or Q).

. Repeats the previous command.

The commands take effect immediately. It is not necessary to enter a carriage return. Up to the time when the command character itself is given, the user may enter the line kill character to cancel the numerical argument being formed. In addition, the user may enter the erase character to redisplay the "--More--(*xx%*)" message.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you enter will not show on your terminal, except for the slash (/) and exclamation (!) commands.

If the standard output is not a teletype, *more* acts just like *cat,* except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

Files

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

See Also

csh(C), sh(C), environ(M)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

The *vi* and *help* options may not be available.

Before displaying a file, *more* attempts to detect whether it is a non-printable binary file such as a directory or executable binary image. If *more* concludes that a file is unprintable, it refuses to print it. However, *more* cannot detect all possible kinds of non-printable files.

mv

moves or renames files and directories

Syntax

mv [**-f**] file1 file2

mv [**-f**] directory1 directory2

mv [**-f**] file ... directory

Description

mv moves (changes the name of) *file1* to *file2* (or *directory1* to *directory2).*

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(S)) and reads the standard input to obtain a line. If the line begins with **y**, the move takes place; if not, *mv* exits.

In the third form, one or more *files* are moved to the *directory* with their original filenames.

No questions are asked when the **-f** option is given.

mv refuses to move a file onto itself.

mv can only rename directories, not physically move them. *mvdir*(ADM) should be used to move directories within a filesystem.

See Also

cp(C), chmod(S), copy(C)

Notes

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Standards Conformance

mv is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

newform

changes the format of a text file

Syntax

newform [-s] [-itabspec] [-otabspec] [-b*n*] [-e*n*] [-p*n*] [-a*n*] [-f] [-c*char*] [-l*n*] [file ...]

Description

newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **-s**, command line options may appear in any order, may be repeated, and may be intermingled with *files*. However, note that command line options are processed in the order typed. This means that option sequences like "-e15 -l60" will yield results different from "-l60 -e15". Options are applied to all *files* on the command line.

-i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described below. In addition, *tabspec* may be **--**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** expects no tabs; if any are found, they are treated as **-1**.

-o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **-i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** means that no spaces will be converted to tabs on output.

-l*n* Sets the effective line length to *n* characters. If *n* is not typed, **-l** defaults to 72. The default line length without the **-l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **-i** to expand tabs to spaces).

Note that the **-l** option used alone does not produce the expected output unless accompanied by other line-altering options, such as **-e**.

-b*n* Truncates *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-l*n***). The default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **-b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

newform -l1 -b7 file-name

The option **-l1** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated.

-e*n* Truncates *n* characters from the end of the line.

-c*k* Changes the prefix/append character to *k*. Default character for *k* is a space (see options -p and -c).

-p*n* Prefixes *n* characters (see **-c*k***) to the beginning of a line when the line length is less than the effective line length. The default is to prefix the number of characters necessary to obtain the effective line length.

-a*n* Appends *n* characters to the end of a line. The default is to append the number of characters necessary to get the effective line length.

-f Writes the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **-o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

Tabs

Four types of tab specification are accepted for *tabspec*: ''canned,'' repetitive, arbitrary, and file. The lowest column number is 1. For *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g. the DASI 300, DASI 300S, and DASI 450.

The ''canned'' tabs are given as *-code* where *code* (and its meaning) is from the following list:

-a 1,10,16,36,72
Assembler, IBM S/370, first format

-a2 1,10,16,40,72
Assembler, IBM S/370, second format

-c 1,8,12,16,20,55
COBOL, normal format

-c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than COBOL **-c2**. This is the recommended format for COBOL. The appropriate format specification is:

<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,53,57,61
PL/I

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these ''canned'' formats, three other types exist:

-n A repetitive specification requests tabs at columns 1+*n*, 1+2**n*, etc. Note that such a setting leaves a left margin of *n* columns on TermiNet terminals *only*. Of particular

importance is the value **-8**: this represents the Altos UNIX System V system "standard" tab setting, and is the most likely tab setting to found at a terminal. It is required for use with *nroff* **-h** option for high-speed output. Another special case is the value **-0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of number, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

- *-file*

If the name of a file is given, *newform* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings.

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-T*type*

newform usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term*(CT). If no **-T** flag is supplied, *newform* searches for the **$TERM** value in the *environment* (see *environ*(M)). If no *type* can be found, *newform* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n*+1 the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Example

In the following example, *newform* converts a file named *text* with leading digits, one or more tabs, and text on each line to a file beginning with the text and the leading digits placed at the end of each line in column 73 (**-s** option). All tabs after the first one are expanded to spaces (**-i** option). To reach the line length of 72 characters (**-l** option), spaces are appended to each line up to column 72 (**-a** option) or lines are truncated at column 72 (-e option). To reformat the sample file **text** in this manner, enter:

```
newform -s -i -l -a -e text
```

Exit Codes

0 - normal execution
1 - for any error

See Also

csplit(C)

Diagnostics

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not -s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

Notes

newform normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i,--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was "-o--" , and was preceded by either "-o--" or a "-i--" , the tab specification format line will be incorrect.

newgrp

logs user into a new group

Syntax

newgrp [-] GROUP

Description

newgrp changes the group identification of its caller. The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

newgrp without an argument changes the group identification to the group in the password file. This changes the caller's group identification back to the original group.

If the first argument to *newgrp* is a hyphen (-), the user will actually be logged in again as a member of the new group, **GROUP**. (that is, *newgrp* - **GROUP**)

If the first argument to *newgrp* is a " - ," but **GROUP** is not specified, the user will be logged in again as a member of the caller's original group identification according to the password file.

Files

/etc/group

/etc/passwd

See Also

login(M), group(F)

Notes

The *newgrp* command executes, but does not fork, a new shell. If your login shell is a C shell and you invoke *newgrp* , you will have to press CTRL-D when you wish to log out. Typing the *csh* (C) *logout* command will result in an error message. Note also that the *newgrp* command causes the *csh* history list to start again at 1.

Standards Conformance

newgrp is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

news

print news items

Syntax

news [**-a**] [**-n**] [**-s**] [items]

Description

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news.**

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

The **-a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The **-n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The -s option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time.

All other arguments are assumed to be specific news items that are to be printed.

If the INTERRUPT key is struck during the printing of a news item, printing stops and the next item is started. Another INTERRUPT within one second of the first causes the program to terminate.

Files

/usr/news/*
$HOME/.news_time

See Also

profile(M), environ(M)

Standards Conformance

news is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

nice

runs a command at a different scheduling priority

Syntax

nice [-increment] command [arguments]

Description

The *nice* command is used to execute a command at a different scheduling priority than usual. Each process has a "nice value" which is used to calculate its priority. Nice values range from 0 to 39, with higher nice values resulting in lower priorities. By default, commands have a nice value of 20. *nice* executes *command* with a nice value equal to 20 plus *increment*. If no *increment* is given, an *increment* of 10 is assumed.

The super-user may run commands with priority *higher* than normal by using a double negative increment. For example, an argument of **--10** would decrement the default to produce a nice value of 10, which is a higher scheduling priority than the default of 20.

See Also

nohup(C), csh(C), nice(S)

Diagnostics

nice returns the exit status of *command*.

Notes

If the default nice value plus *increment* is larger than 39, a nice value of 39 will be used. If a nice value less than zero is requested, zero will be used.

Note also that this description of *nice* applies only to programs run under the Bourne Shell. The C-Shell has its own *nice* command, which is documented in csh(C).

Standards Conformance

nice is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

nl

adds line numbers to a file

Syntax

nl [**-h**type] [**-b**type] [**-f**type] [**-v**start#] [**-i**incr] [**-p**] [**-l**num] [**-s**sep] [**-w**width] [**-n**format] file

Description

nl reads lines from the named *file*, or the standard input if no *file* is named, and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g. no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections is signaled by input lines containing nothing but the following character(s):

Page Section	*Line Contents*
Header	\:\:\:
Body	\:\:
Footer	\:

Unless signaled otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional filename. Only one file may be named. The options are:

-b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p***string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).

-h*type* Same as **-b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

-f*type* Same as **-b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

-p Does not restart numbering at logical page delimiters.

-v*start#* *Start#* is the initial value used to number logical page lines. Default is **1**.

-i*incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

-s*sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

-w*width* *Width* is the number of characters to be used for the line number. Default *width* is **6**.

-n*format* *Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

-l*num* *Num* is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). Default is **1**.

See Also

pr(C)

Standards Conformance

nl is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

nohup

runs a command immune to hangups and quits

Syntax

nohup command [arguments]

Description

nohup executes *command* with hangups and quits ignored. If output is not redirected by the user, it will be sent to **nohup.out**. If the user does not have write permission in the current directory, output is redirected to **$HOME/nohup.out**.

See Also

nice(C), signal(S)

Standards Conformance

nohup is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

od

displays files in octal format

Syntax

od [-bcdox] [file] [[+]offset[**.**][**b**]]

Description

od displays *file* in one or more formats as selected by the first argument. If the first argument is missing, **-o** is default. The meanings of the format options are:

-b Interprets bytes in octal.

-c Interprets bytes in ASCII. Certain nongraphic characters appear as C escapes: null=**\0**, backspace=**\b**, form feed=**\f**, newline=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.

-d Interprets words in decimal.

-o Interprets words in octal.

-x Interprets words in hex.

The *file* argument specifies which file is to be displayed. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where displaying is to start. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks. If the file argument is omitted, the offset argument must be preceded by +.

The display continues until end-of-file.

See Also

hd(C), adb(CP)

Standards Conformance

od is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

and The X/Open Portability Guide II of January 1987.

otar

original tape archive command

Syntax

otar

Syntax

tar [crtux] [bBefFiIhklmnopsvVw 0,...,7] [arguments] file ...

Description

The *otar* command saves and restores files on magnetic tape or floppy disk. Altos UNIX System V contains *otar* to provide compatibility with original *tar* archives made on the earlier Altos System V operating system. Refer to the "Notes" section below for other guidelines on using *tar* to read archives produced with *otar* (or with *tar* on Altos System V).

In addition to providing compatability with Altos System V's original *tar* command, *otar* offers these other features:

- Creates/copies directories even if they are empty, whereas *tar* does not.
- Copies device nodes and pipes, whereas *tar* does not.
- Retains file and directory permissions as originally copied (unless when used with the **o** option).
- Incremental backups (with the **I** option).
- Multivolume archiving.

The actions produced by *otar* are controlled by a key argument, which contains at least one function letter followed by one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped or restored. In all cases, a directory name refers to the files and (recursively) subdirectories of that directory.

Tar permits a file to extend across media boundaries.

Specify the function portion of the key by one of the following letters:

c Creates a new tape; writing begins at the beginning of the tape instead of after the last file. When you use this command, all previous data is erased.

r Writes the named files at the end of the tape (only for seekable devices).

t Lists the named file each time it occurs on the tape. If no file argument is given, all of the names on the tape are listed.

u Adds the named file to the tape if it is not already there or if it has been modified since last put on the tape. This option can be slow (only for seekable devices).

x Extracts the named file from the tape. If the named file matches a directory whose contents have been written on the tape, this directory is (recursively) extracted. The owner and mode are restored (if possible). If no file argument is given, the entire content of the tape or floppy is extracted. If multiple entries specifying the same file are on the tape, the last version will overwrite all preceding versions.

In addition to the key argument function, you can use the following modifiers. Arguments to the modifiers are given in the same order as the modifiers themselves.

b Causes *otar* to use the next argument as the blocking factor for tape records. The default blocking size is 18 for floppies, 126 for cartridge tapes, 20 for other tapes, and 1 for all other devices. Use the same blocking factor on the **x** (extract) as used on the **c** (create) option. (The *tar* default for Altos System V on 386 computers was also 18, and the maximum was 1024. The blocking defaults for the Altos UNIX System V *tar* are set in **/etc/default/tar**.)

This option should be used to set different blocking factors only on raw magnetic tape archives (see **f** below). Use the default blocking factors for all other devices.

Don't use the **b** option with archives that are going to be updated. If the archive is on a disk file, the **b** option should not be used at all, as updating an archive stored in this manner can destroy it.

B Archives all files modified after the modification date and time of the file you specify (instead of **/etc/bkupdate**).

Can only be used with the **I** option. Also *otar* sets the modification time of the given file after the backup is complete. The **B** option sets the modification time in the user-specified file. For example:

otar cvfbBI /dev/rct0 1024 /etc/timefile ./*

The user-specified file is set to zero length when its modification date is set.

e Prevents files from being split across volumes (tapes or disks). If there is not enough room on the present volume for a given file, *otar* prompts for a new volume. This is only valid when you also specify the **k** option.

f Causes *otar* to use the next argument as the name of the archive instead of **/dev/tar**. If the name of the file is '-', *otar* writes to standard output or reads from standard input, whichever is appropriate. Thus, you can use tar to move hierarchies with the command:

cd fromdir; otar cf - . | (cd todir; otar xf -)

You must use this option with magnetic tape and add-on hard disks. The default is to floppy disk.

F Causes *otar* to use the next argument as the name of a file from which succeeding arguments are taken. A dash (-) signifies that arguments are taken from the standard input.

h Archives the contents of the symbolically-linked named files. *otar cv* will only archive linkage information; *tar chv* will archive the contents.

i *date time*
Archives all files modified after *date* and *time*. The format for *date* and *time* is:

MM/DD/YY,HH:MIN:SEC

Files modified before *date* and *time* will be skipped. Any trailing portion may be omitted. *DD*, *HH*, and *MIN* default to 0; *YY* defaults to the current year. For example:

otar cvif 12/22/86,04:00:00 /dev/rct0 files

I Archives all files modified after the date and time as defined by the modification time of the file **/etc/bkupdate**. Also, sets the modification time of **/etc/bkupdate** after the backup is complete. To use a different file, see the **B** option.

k Causes *otar* to use the next argument as the size of an archive volume in kilobytes. The minimum value allowed is 250. This value must be a multiple of the blocking factor (9K by default). For tape, you can specify the block size using the **b** option. Very large files are split into "extents" across volumes. When restoring from a multivolume archive, *otar* only prompts for a new volume if a split file has been partially restored.

l Tells *otar* to notify you if the link count of a dumped file doesn't match the actual number of dumped links to that file. If this option is not specified, no error messages are printed.

m Tells *otar* not to restore the modification time; the time of extraction then becomes the modification time.

n Indicates the archive device is not a magnetic tape. The **k** option implies this. Because it can seek over files it wishes to skip, *otar* can quickly list and extract the contents of an archive. Sizes are printed in kilobytes instead of tape blocks.

o Causes extracted files to take on the user/group identifier of the user running the program, rather than those on the tape.

p Indicates that files are extracted using their original permissions. It is possible that a regular user may be unable to extract files because of the permission associated with the files or directories being extracted.

s *file*
Runs the */bin/sum* algorithm on the archive and writes the resulting checksum in file.

v Displays the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name and path.

V Verifies the named file on the tape. *otar* will compare the tape file to the disk file and report any file change or comparison errors. If no file argument is given, the entire contents of the tape or floppy is verified. *otar* will exit with an exit code of 9 if there are any verify errors.

w Causes *otar* to display the action to be taken and file name, then wait for user confirmation. If you type **y**, the action is performed. Any other input causes the file to be skipped.

0,...,7
Selects the drive on which the archive is mounted. This option should only be selected if you have linked the appropriate **/dev/mt** to the desired device.

Files

/dev/tar	Default input/output device
/tmp/tar*	

Examples

This command copies the directory **/usr/john** to floppy disk(s):

```
otar cv /usr/john
```

This command copies the files on the floppy disk to the directory **/usr/john**. The *cd* command is used first to make sure you are in the correct directory:

```
cd /usr/john
otar xv
```

This command displays the contents of the floppy disk you have in the drive:

```
otar tv
```

This command pipes the **otar tv** command through the **lpr** command. This causes the contents of the floppy disk to be printed out on your serial printer:

```
otar tv | lpr
```

This command copies files from a floppy disk device named **/dev/fd196ds15**, a 5¼ inch floppy drive configured as the second floppy drive (fd1). (Other arguments are **files**, the names of files to archive, and **1152**, the capacity of the disk in kilobytes). Arguments to key letters are given in the same order as the key letters themselves, thus the **fk** key letters have corresponding arguments **/dev/fd196ds15** and **1152**. If a file is a directory, the contents of the directory are recursively archived:

```
otar cvfk /dev/fd196ds15 1152 files
```

This command extracts all the files with the exact same pathnames used when the archive was created:

```
otar xvf /dev/fd096ds15
```

This command copies the directory **/usr/john** to cartridge tape(s):

```
otar cvfb /dev/rct0 126 /usr/john
```

Notes

If you use *tar* to read an archive created by *otar*, you may see error messages concerning directories, pipes, or device nodes. You should ignore these messages. Your *tar* operation will still work. Note, however, that you should still use the same blocking factor to read a tape as was used to originally create the tape.

Value Added

otar is an extension of AT&T System V provided by Altos UNIX System V.

pack, pcat, unpack

compresses and expands files

Syntax

pack [-] name ...

pcat name ...

unpack name ...

Description

pack attempts to store the specified files in a compressed form. Wherever possible, each input file *name* is replaced by a packed file *name*.**z** with the same access modes, access and modified dates, and the owner of *name*. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes *pack* to display information about the file compression. Additional occurrences of - in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very scattered, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

- The file appears to be already packed

- The filename has more than 12 characters
- The file has links
- The file is a directory
- The file cannot be opened
- No disk storage blocks will be saved by packing
- A file called *name*.**z** already exists
- The .**z** file cannot be created
- An I/O error occurred during processing

The last segment of the filename must contain no more than 12 characters to allow space for the appended .**z** extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(C) does for ordinary files. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.**z** use:

pcat name.z

or just:

pcat name

To make an unpacked copy, say *nnn*, of a packed file named *name*.**z** without destroying *name*.**z**, enter the command:

pcat name >nnn

Pcat returns the number of files it was unable to unpack. Failure may occur if:

- The filename (exclusive of the .**z**) has more than 12 characters
- The file cannot be opened
- The file does not appear to be the output of *pack*

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.**z** (or just *name*, if *name* ends in .**z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .**z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as in a file where the "unpacked" name already exists, or if the unpacked file cannot be created.

Standards Conformance

pack, *pcat* and *unpack* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

passwd

change login, modem (dialup shell), filesystem, or group password

Syntax

passwd [**-mgF**] [**-dluf**] [**-n** minimum] [**-x** expiration] [**-r** retries] [name]
passwd -s [**-a**] [name]

Description

The *passwd* command is used by ordinary users to:

- Change or delete their own login password.
- List some of the attributes that apply to their account.

In addition, system administrators can use the *passwd* command to:

- Change or delete any user's login password.
- Change or delete modem (dialup shell), filesystem mount, and group passwords.
- Lock or unlock any user's account.
- Invalidate (lock) dialup shell, filesystem, and group passwords.
- List some of the attributes of all users, or any single user.
- Change some of the attributes of any user.

However, it is recommended that system administrators use the *sysadmsh*(ADM) **Accounts** selection to administrate passwords. A user is considered to be a system administrator if they are logged in as someone who has the **auth** subsystem authorization.

Choosing a good password.

Your login password is one of the most important defenses against security breaches. If a malicious person cannot log into a system, it is much harder for that person to steal or tamper with your data. Hence, by choosing a hard-to-guess password (either of your own invention or one suggested by the system), regularly changing it, and keeping it secret, you can foil many attacks on your system.

In general, a password *should*:

- Consist of a mixture of upper- and lower-case letters, digits (0 - 9), and other non-letters (such as @, *, -, /, space, tab, and control characters).
- Be changed frequently (at least once every six months to a year, and more often as necessary).
- Be different on different machines.
- Be easy to remember, so you don't have to write it down.
- Be kept secret and known only by you.

Passwords *should not*:

- Be the name of a person, place, or thing; nor should a password be the same as any user's login name, any machine's name, or the name of any group.
- Be a correctly spelt word, street or telephone number, ZIP or postal code; nor should a password be a birthday or anniversary of you or anyone you know.
- Be written down (anywhere! - not on paper or in a file); nor should passwords be stored in the function keys of a terminal or memory of an intelligent modem.
- Be told to any other person (not even for use in an "emergency"); nor should a password be kept if you suspect someone else knows it.

Spelling a word backwards or appending a digit to a word do *not* turn a poor password choice into a "good" password. However, taking two or three unrelated words and combining them with some non-letters is a reasonable way of choosing an easy-to-remember but hard-to-crack password. On Altos UNIX System V, passwords can be up to 80 characters long, so nonsensical rhymes (for example) can also be used as passwords.

User login passwords.

When *passwd* is used to change or delete the password for user *name*, the old password (if any) is prompted for. (The password is not displayed as it is being entered.) System administrators are not prompted for the old password unless they are attempting to change their own password; the superuser is never prompted for the old password. The *passwd* command can only be used to change or delete the password for user *name* by system administrators and the user authorized to change user *name*'s password. Normally, users are authorized to change their own password.

Depending on how the system administrator has configured the account, the user may or may not be able to choose their own password, or may have a password chosen for them. If they can neither choose their own password nor have passwords generated for them, the password cannot be changed. If the user is able to do both, *passwd* asks which should be done.

A password is considered *valid* until it has *expired*. Passwords expire if they are not changed or deleted before the expiration time has passed. Once expired, the user is required to change (not delete) their password the next time they log in. If a user fails to do so before the password's lifetime has passed, the password is considered *dead* and the user's account is *locked*.

Once locked, the user may not log in, may not be *su*(C)'ed to, and no *at*(C), *batch*(C), or *cron*(C) jobs for that user may run. Only a system administrator can unlock a user with a dead password; a new password must be assigned.

To discourage re-use of the same password, the system administrator may set a *minimum change time*. After changing or deleting a password, the password may not be changed again (even by a system administrator) until at least that much time has elapsed.

Passwords may be deleted (or changed to be empty) only if the user is authorized to not have a password. Users without passwords are not recommended. (An empty password is prompted for when logging in, but a deleted password is not prompted for at login.)

If a password is being changed and the user has elected (or is forced) to choose a system-generated password, each suggested password is printed along with a hyphenated spelling that suggests how the password could be pronounced. To accept a suggested password, enter the password; if entered correctly, *passwd* will prompt for the suggested password to be entered again as confirmation. To reject a suggestion, just enter RETURN ; to abort the change altogether, either enter "quit" or interrupt *passwd*.

If a password is being changed and the user has elected (or is forced) to assign a password of their own choosing, the new password is prompted for twice. It is checked for being "obvious" after the first prompt, and if deemed to be acceptable is prompted for again. If the proposed password is successfully entered a second time, it becomes the new password for user *name*.

Both system-generated and self-chosen passwords are checked for being easy-to-guess. See the section on "Checking for obvious passwords" (below) for a description of the checks.

When dealing with a user's login password, the following options are recognized:

-d Delete the password. A password may be deleted only if the user is authorized to not have a password. System administrators must always specify *name*; otherwise, the name of the user who logged in is used.

-f Force user *name* to change their password the next time they log in. This option may be specified only by system administrators, and only when the user's password is not being changed or deleted; *name* must be explicitly given.

-l Lock user *name* out of the system by applying an administrative lock; only system administrators may do this and they must specify *name*.

-u Remove any administrative lock applied to user *name*; only system administrators may do this and they must specify *name*.

-n *minimum*
Set the amount of time which must elapse between password changes for user *name* to *minimum* days. Only system administrators may do this and they must specify *name*.

-x *expiration*
Set the amount of time which may elapse before the password of user *name* expires to *expiration* days. Only system administrators may do this and they must specify *name*. Once a password has expired, the user must change it the next time they log in.

-r *retries*
Up to *retries* attempts may be made to choose a new password for user *name*.

-s Report the password attributes of user *name* (or, if the **-a** option is given, of all users). The format of the report is:

name status mm/dd/yy minimum expiration

where *status* is **PS** if the user has a password, **LK** is the user is administratively locked, or **NP** when the user does not have a password. The date of the last successful password change (or deletion) is shown as *mm/dd/yy*. If neither *name* nor **-a** is specified, the name of the user who logged in is assumed. Only system administrators can examine the attributes of users other than themselves.

If no **-d**, **-f**, **-l**, **-u**, or **-s** option is specified, the password for user *name* is changed as described above. If no *name* is given and no option which requires *name* is given, then the *name* of the user who logged in is used. Only the **-a** option may be specified with the **-s** option.

Modem (dialup shell) passwords.

When a user whose login shell is listed in **/etc/d_passwd** with a (encrypted) password logs in on a terminal line listed in **/etc/dialups**, the password in **/etc/d_passwd** must be supplied before the login succeeds. The **-m** option to password allows system administrators to change, delete, or invalidate (lock) the passwords for login shell *name*:

-d Delete the password.

-l Invalidate ("lock") the password by arranging so that no matter what the user enters, it will not be a valid password. Doing so causes the old password to be lost.

-r *retries*
Up to *retries* attempts may be made to choose a new password.

The *name* must always be specified. If *name* begins with a slash ("/") the entire shell pathname must match. Otherwise the password for every shell whose basename is *name* is changed.

If neither the **-d** nor **-l** option is specified, the password is changed. The new password is prompted for twice, and must pass checks similar to those for login passwords (see below).

Filesystem mount passwords.

A password may be required when mounting a filesystem; see *mnt*(C). The **-F** option to *passwd* allows system administrators to change, delete, or invalidate (lock) the password for filesystem *name*. The options are the same as for modem passwords (see above).

Group passwords.

A password may be required when a user changes their current working group; see *newgrp*(C). The **-g** option to *passwd* allows system administrators to change, delete, or invalidate (lock) the password for group *name*. The options are the same as for modem passwords (see above).

Checking for obvious passwords.

To discourage poor password choices, various checks are applied to reject unacceptable passwords. The checks which are applied depend on the type of password being checked and the system's configuration. Most of the checks for being easy-to-guess are configurable; see *goodpw*(ADM).

The check procedure is as follows (a password is *restricted* if, according to *sysadmsh* **Accounts**, it is to be "checked for obviousness"):

1a.
User login passwords only: The new password must not be the same as the old password. The password must not be empty (or be deleted) unless the user is not required to have a password.

1b.
All other passwords: The new and old password may be the same. Empty passwords are treated as deleted passwords and are always acceptable.

2. All (non-empty) passwords: If the password is not empty, it must be at least **PASSLENGTH** characters long (see below).

3. All (non-empty) passwords: If the *goodpw* utility can be run, it is used to perform all further checks. If the file *CHECKDIR/type/strength* exists (and can be read by *goodpw*) that file is used to modify the default settings in **/etc/default/goodpw**. The *CHECKDIR* is specified by **CHECKDIR** in **/etc/default/passwd** and *type* is the kind of password being checked (**user**, **modem**, **group**, or **filsys**). The *strength* is the degree of checking to be done: **secure** if the user is restricted (or, for all other password types, if the system default is restricted); otherwise **weak**.

4. When *goodpw* cannot be run (all passwords): If the password is not empty, it must contain at least one character which is not a lower case letter (but must not consist solely of digits).

5. When *goodpw* cannot be run (user login passwords only): Finally, for user login passwords which are restricted, the password must not be a palindrome, any user's login name, the name of any group, or a correctly spelt English word (American spelling); see *accept_pw*(S).

System-generated passwords are not checked unless the user is restricted (see above), in which case the generated password must pass the checks in step *5* before it is suggested to the user. Generated passwords are never checked by *goodpw*. The minimum value for **PASSLENGTH**, and the minimum length of a generated password, are computed based on the password's lifetime, delay between login attempts, and other factors; see *passlen*(S).

Defaults.

Several parameters may be specified in **/etc/default/passwd**. The various settings, and their default values are:

PASSLENGTH=5
: The minimum length of a password. If outside the range 3 to 80 (inclusive), then it is set to 5. The actual minimum length used by *passwd* is the maximum of this value and a value computed by taking into consideration the lifetime of the password (and other factors).

RETRIES=4
: The maximum number of repeated attempts to change a password that has been rejected. If less than 2, then 2 is assumed.

ONETRY=YES
: If set to **YES**, a rejected password is added to the stop-list passed to *goodpw*. This prevents simplistic modifications of a rejected password from being accepted on a later attempt.

DESCRIBE=/usr/lib/goodpw/describe
: The contents of this file are shown once (before the new password is prompted for) and should describe the the difference between acceptable and unacceptable passwords.

SUMMARY=/usr/lib/goodpw/summary
: The contents of this file are shown each time a password is rejected, and should be a (short) reminder of what are and are not acceptable passwords.

CHECKDIR=/usr/lib/goodpw/checks
: A hierarchy of additional checks *goodpw* should perform, based on password type and restrictions (see above).

GOODPW=/usr/bin/goodpw
: An independent program that applies various checks in an attempt to determine whether or not a password is easily guessed.

The values for the default settings may be changed to reflect the system's security concerns.

If **/etc/default/passwd** does not exist or is not readable, the above default values are used.

If the **DESCRIBE** or **SUMMARY** file defined in **/etc/default/passwd** does not exist or cannot be read, short (and vague) descriptions or summaries are issued instead. In addition, if the user who logged in is a system administrator, an error message describing the problem is printed.

If the **GOODPW** program does not exist or is not executable, simpler checks are done (see above). In addition, if the user who logged in is a system administrator, an error message describing the problem is printed.

Files

/etc/passwd
List of user accounts.

/tcb/files/auth/*initial***/***name*
Protected Password database entry for user *name* (where the first character in *name* is *initial*).

/etc/group
List of groups.

/etc/d_passwd
List of dialup shells and passwords (one per line):

shell:*encrypted-password*:*reserved*

where *shell* is the pathname of a login shell as used in **/etc/passwd**.

/etc/auth/system/files
File Control database.

/etc/auth/system/default
System Defaults database; contains default parameters.

/etc/default/passwd
Configurable settings (see above).

See Also

accept_pw(S), authcap(F), authsh(ADM), default(F), goodpw(ADM), group(F), login(M), mnt(C), newgrp(C), passlen(S), passwd(F)

Notes

Group passwords should be avoided; see *newgrp*(C). Not all systems support group passwords.

Not all systems support filesystem mount passwords.

Not all systems support modem (dialup shell) passwords.

The **-r** option is mostly useful during installation to force the newly-installed superuser to have a password.

paste

merges lines of files

Syntax

paste file1 file2 ...

paste -dle1 file2 ...

paste -s [-dlist] file1 file2 ...

Description

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of *cat*(C) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if **-** is used in place of a filename.

The meanings of the options are:

-d Without this option, the newline characters of each but the last file (or last line in case of the **-s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters. (See below.)

list
One or more characters immediately following **-d** replace the default *tab* as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no **-s** option), the lines from the last file are always terminated with a newline character, not from the *list*. The list may contain the special escape sequences: **\n** (newline), **\t** (tab), **** (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use *-d*"\\\\").

-s Merges subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **-d** option. Regardless of the *list*, the very last character of the file is forced to be a newline.

- May be used in place of any filename to read a line from the standard input. (There is no prompting.)

Examples

ls | paste -d" " -	Lists directory in one column
ls | paste - - - -	Lists directory in four columns
paste -s -d"\t\n" file	Combines pairs of lines into lines

See Also

cut(C), grep(C), pr(C)

Diagnostics

line too long	Output lines are restricted to 511 characters.
too many files	Except for -s option, no more than 12 input files may be specified.

Standards Conformance

paste is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

pax

portable archive exchange

Syntax

pax [**-cimopuvy**] [**-f** *archive*] [**-s** *replstr*] [**-t** *device*] [*pattern...*]

pax -r [**-cimnopuvy**] [**-f** *archive*] [**-s** *replstr*] [**-t** *device*] [*pattern...*]

pax -w [**-adimuvy**] [**-b** *blocking*] [**-f** *archive*] [**-s** *replstr*] [**-t** *device*] [**-x** *format*] [*pathname...*]

pax -rw [**-ilmopuvy**] [**-s** *replstr*] [*pathname...*] directory

Description

pax reads and writes archive files which conform to the **Archive/Interchange File Format** specified in *IEEE Std. 1003.1-1988*. *pax* can also read, but not write, a number of other file formats in addition to those specified in the **Archive/Interchange File Format** description. Support for these traditional file formats, such as V7 *tar* and System V binary *cpio* format archives, is provided for backward compatibility and to maximize portability.

pax will also support traditional *cpio* and System V *tar* interfaces if invoked with the name "cpio" or "tar" respectively. See the *cpio*(C) or *tar*(C) manual pages for more details.

Combinations of the **-r** and **-w** command line arguments specify whether *pax* will read, write or list the contents of the specified archive, or move the specified files to another directory.

The command line arguments are:

-w writes the files and directories specified by *pathname* operands to the standard output together with the pathname and status information prescribed by the archive format used. A directory *pathname* operand refers to the files and (recursively) subdirectories of that directory. If no *pathname* operands are given, then the standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied.

-r *pax* reads an archive file from the standard input. Only files with names that match any of the *pattern* operands are selected for extraction. The selected files are conditionally created and copied relative to the current directory tree, subject to the options described below. By default, the owner and group of selected files will be that of the invoking process, and the permissions and modification times will be the sames as those in the archive.

The supported archive formats are automatically detected on input. The default output format is *ustar*, but may be overridden by the **-x** *format* option described below.

-rw *pax* reads the files and directories named in the *pathname* operands and copies them to the destination *directory*. A directory *pathname* operand refers to the files and (recursively) subdirectories of that directory. If no *pathname* operands are given, the standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied. The directory named by the *directory* operand must exist and have the proper permissions before the copy can occur.

If neither the **-r** or **-w** options are given, then *pax* will list the contents of the specified archive. In this mode, *pax* lists normal files one per line, hard link pathnames as

pathname == linkname

and symbolic link pathnames (if supported by the implementation) as

pathname -> linkname

where *pathname* is the name of the file being extracted, and *linkname* is the name of a file which appeared earlier in the archive.

If the **-v** option is specified, then *pax* list normal pathnames in the same format used by the *ls* utility with the **-l** option. Hard links are shown as

<ls -l listing> == linkname

and symbolic links (if supported) are shown as

<ls -l listing> -> linkname

pax is capable of reading and writing archives which span multiple physical volumes. Upon detecting an end of medium on an archive which is not yet completed, *pax* will prompt the user for the next volume of the archive and will allow the user to specify the location of the next volume.

Options

The following options are available:

-a The files specified by *pathname* are appended to the specified archive.

-b *blocking* Block the output at *blocking* bytes per write to the archive file. A **k** suffix multiplies *blocking* by 1024, a **b** suffix multiplies *blocking* by 512 and a **m** suffix multiplies *blocking* by 1048576 (1 megabyte). If not specified, *blocking* is automatically determined on input and is ignored for **-rw.**

-c Complement the match sense of the the *pattern* operands.

-d Intermediate directories not explicitly listed in the archive are not created. This option is ignored unless the **-r** option is specified.

-f *archive* The *archive* option specifies the pathname of the input or output archive, overriding the default of standard input for **-r** or standard output for **-w**.

-i Interactively rename files. Substitutions specified by **-s** options (described below) are performed before requesting the new file name from the user. A file is skipped if an empty line is entered and *pax* exits with an exit status of 0 if **EOF** is encountered.

-l Files are linked rather than copied when possible.

-m File modification times are not retained.

-n When **-r** is specified, but -w is not, the *pattern* arguments are treated as ordinary file names. Only the first occurrence of each of these files in the input archive is read. The **pax** utility exits with a zero exit status after all files in the list have been read. If one or more files in the list is not found, **pax** writes a diagnostic to standard error for each of the files and exits with a non-zero exit status. the file names are compared before any of the **-i**, **-s**, or **-y** options are applied.

-o Restore file ownership as specified in the archive. The invoking process must have appropriate privileges to accomplish this.

-p Preserve the access time of the input files after they have been copied.

-s *replstr* File names are modified according to the substitution expression using the syntax of *ed(C)* as shown:

-s */old/new/*[gp]

Any non null character may be used as a delimiter (a / is used here as an example). Multiple **-s** expressions may be specified; the expressions are applied in the order specified terminating with the first successful substitution. The optional trailing **p** causes successful mappings to be listed on standard error. The optional trailing **g** causes the *old* expression to be replaced each time it occurs in the source string. Files that substitute to an empty string are ignored both on input and output.

-t *device* The *device* option argument is an implementation-defined identifier that names the input or output archive device, overriding the default of standard input for **-r** and standard output for **-w**.

-u Copy each file only if it is newer than a pre-existing file with the same name. This implies **-a**.

-v List file names as they are encountered. Produces a verbose table of contents listing on the standard output when both **-r** and **-w** are omitted, otherwise the file names are printed to standard error as they are encountered in the archive.

-x *format* Specifies the output archive *format*. The input format, which must be one of the following, is automatically determined when the **-r** option is used. The supported formats are:

cpio The extended *CPIO* interchange format specified in **Extended CPIO Format in** *IEEE Std. 1003.1-1988.*

ustar The extended *TAR* interchange format specified in **Extended TAR Format in** *IEEE Std. 1003.1-1988.* This is the default archive format.

-y Interactively prompt for the disposition of each file. Substitutions specified by **-s** options (described above) are performed before prompting the user for disposition. **EOF** or an input line starting with the character **q** caused *pax* to exit. Otherwise, an input line starting with anything other than **y** causes the file to be ignored. This option cannot be used in conjunction with the **-i** option.

Only the last of multiple **-f** or **-t** options take effect.

When writing to an archive, the standard input is used as a list of pathnames if no *pathname* operands are specified. The format is one pathname per line. Otherwise, the standard input is the archive file, which is formatted according to one of the specifications in **Archive/Interchange File format** in *IEEE Std. 1003.1-1988*, or some other implementation-defined format.

The user ID and group ID of the process, together with the appropriate privileges, affect the ability of *pax* to restore ownership and permissions attributes of the archived files. (See *format-reading utility* in **Archive/Interchange File Format** in *IEEE Std. 1003.1-1988*.)

The options **-a, -c, -d, -i, -l, -p, -t, -u,** and **-y** are provided for functional compatibility with the historical *cpio* and *tar* utilities. The option defaults were chosen based on the most common usage of these options, therefore, some of the options have meanings different than those of the historical commands.

Operands

The following operands are available:

directory The destination directory pathname for copies when both the **-r** and -**w** options are specified. The directory must exist and be writable before the copy or and error results.

pathname A file whose contents are used instead of the files named on the standard input. When a directory is named, all of its files and (recursively) subdirectories are copied as well.

pattern A *pattern* is given in the standard shell pattern matching notation. The default if no *pattern* is specified is *, which selects all files.

Examples

The following command

```
pax -w -f /dev/rmt0 .
```

copies the contents of the current directory to tape drive 0.

The commands

mkdir *newdir*
cd *olddir*

pax -rw . *newdir*

copies the contents of *olddir* to *newdir* .

The command

pax -r -s ',//*usr//*,,' -f pax.out

reads the archive **pax.out** with all files rooted in "/usr" in the archive extracted relative to the current directory.

Files

/dev/tty   used to prompt the user for information when the **-i** or **-y** options are specified.

See Also

cpio(C), find(C), tar(C), cpio(M), tar(F)

Diagnostics

pax will terminate immediately, without processing any additional files on the command line or in the archive.

pax will exit with one of the following values:

0   All files in the archive were processed successfully.

>0   *pax* aborted due to errors encountered during operation.

Notes

Special permissions may be required to copy or extract special files.

Device, user ID, and group ID numbers larger than 65535 cause additional header records to be output. These records are ignored by some historical version of *cpio(C)* and *tar(C)*.

The archive formats described in **Archive/Interchange File Format** have certain restrictions that have been carried over from historical usage. For example, there are restrictions on the length of pathnames stored in the archive.

When getting an ''ls -l'' style listing on *tar* format archives, link counts are listed as zero since the *ustar* archive format does not keep link count information.

Copyright

Author

Mark H. Colburn
NAPS International
117 Mackubin Street, Suite 1
St. Paul, MN 55102
mark@jhereg.MN.ORG

Sponsored by **The USENIX Association** for public distribution.

pcpio

copy file archives in and out

Syntax

pcpio -o[Bacv]
pcpio -i[Bcdfmrtuv] [*pattern...*]
pcpio -p[adlmruv] *directory*

Description

The **pcpio** utility produces and reads files in the format specified by the **cpio Archive/Interchange File Format** specified in *IEEE Std. 1003.1-1988*.

The **pcpio -i** (copy in) utility extracts files from the standard input, which is assumed to be the product of a previous **pcpio -o** . Only files with names that match *patterns* are selected. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is selecting all files. The extracted files are conditionally created and copied into the current directory, and possibly any levels below, based upon the options described below and the permissions of the files will be those of the previous **pcpio -o** . The owner and group of the files will be that of the current user unless the user has appropriate privileges, which causes **pcpio** to retains the owner and group of the files of the previous **pcpio -o** .

The **pcpio -p** (pass) utility reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* based upon the options described below.

If an error is detected, the cause is reported and the **pcpio** utility will continue to copy other files. **pcpio** will skip over any unrecognized files which it encounters in the archive.

The following restrictions apply to the **pcpio** utility:

1 Pathnames are restricted to 256 characters.

2 Appropriate privileges are required to copy special files.

3 Blocks are reported in 512-byte quantities.

Options

The following options are available:

-B Input/output is to be blocked 5120 bytes to the record. Can only be used with **pcpio -o** or **pcpio -i** for data that is directed to or from character special files.

-a Reset access times of input files after they have been copied. When the **-l** option is also specified, the linked files do not have their access times reset. Can only be used with **pcpio -o** or **pcpio -i** .

-c Write header information in ASCII character for for portability. Can only be used with **pcpio -i** or **pcpio -o** . Note that this option should always be used to write portable files.

-d Creates directories as needed. Can only be used with **pcpio -i** or **pcpio -p** .

-f Copy in all files except those in *patterns* . Can only be used with **pcpio -i** .

-l Whenever possible, link files rather than copying them. Can only be used with **pcpio -p** .

-m Retain previous modification times. This option is ineffective on directories that are being copied. Can only be used with **pcpio -i** or **pcpio -p** .

-r Interactively rename files. The user is asked whether to rename *pattern* each invocation. Read and write permissions for **/dev/tty** are required for this option. If the user types a null line, the file is skipped. Should only be used with **pcpio -i** or **pcpio -o** .

-t Print a table of contents of the input. No files are created. Can only be used with **pcpio -i** .

-u Copy files unconditionally; usually an older file will not replace a new file with the same name. Can only be used with **pcpio -i** or **pcpio -p** .

-v Verbose: cause the names of the affected files to be printed. Can only be used with **pcpio -i** . Provides a detailed listing when used with the -t option.

Operands

The following operands are available:

patterns Simple regular expressions given in the name-generating notation of the shell.

directory The destination directory.

Exit Status

The **pcpio** utility exits with one of the following values:

0 All input files were copied.

2 The utility encountered errors in copying or accessing files or directories. An error will be reported for nonexistent files or directories, or permissions that do not allow the user to access the source or target files.

It is important to use the **-depth** option of the **find** utility to generate pathnames for **pcpio** . This eliminates problems **pcpio** could have trying to create files under read-only directories.

The following command:

```
ls | pcpio -o > ../newfile
```

copies out the files listed by the **ls** utility and redirects them to the file **newfile** .

The following command:

```
cat newfile | pcpio -id "memo/al" "memo/b*"
```

uses the output file **newfile** from the **pcpio -o** utility, takes those files that match the patterns **memo/al** and **memo/b*** , creates the directories below the current directory, and places the files in the appropriate directories.

The command

```
find . -depth -print | pcpio -pdlmv newdir
```

takes the file names piped to it from the **find** utility and copies or links those files to another directory named **newdir** , while retaining the modification time.

Files

/dev/tty used to prompt the user for information when the **-i** or **-r** options are specified.

See Also

find(C), pax(C), tar(C), tar(F)

Copyright

Author

Mark H. Colburn
NAPS International
117 Mackubin Street, Suite 1
St. Paul, MN 55102
mark@jhereg.MN.ORG

Sponsored by **The USENIX Association** for public distribution.

Standards Conformance

pcpio is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

pg

file perusal filter for soft-copy terminals

Syntax

pg [- number] **[-p** string] **[-cefns]** [+ linenumber] [+/ pattern /]
[files ...]

Description

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. (The dash (-) command line option and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If you press the RETURN key, another page is displayed; other possibilities are listed below. This command is different from previous paginators because it allows you to back up and review something that has already passed.

To determine terminal attributes, *pg* scans the *termcap*(F) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

-number Specifies the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23.)

-p *string* Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is a colon (:).

-c Homes the cursor and clears the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *termcap*(F) data base.

-e Causes *pg not* to pause at the end of each file.

-f Inhibits *pg* from splitting lines. In the absence of the **-f** option, *pg* splits lines longer than the screen width, but some sequences of characters in the displayed text (for example, escape sequences for underlining) give undesirable results.

-n Normally, commands must be terminated by pressing the RETURN key (ASCII newline character). This option causes an automatic end of command as soon as a command letter is entered.

-s Causes *pg* to display all messages and prompts in standout mode (usually inverse video).

+*linenumber* Starts up at *linenumber*.

+/*pattern*/ Starts up at the first line containing the regular expression pattern.

The responses that may be entered when *pg* pauses can be divided into three categories: those that cause further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address* (an optionally signed number indicating the point from which further text should be displayed). *pg* interprets this *address* in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address if no address is provided.

The perusal commands and their defaults are as follows:

(+1)RETURNkey
Causes one page to be displayed. The *address* is specified in pages.

(+1) **l**
With a signed *address*, causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an unsigned *address* this command displays a full screen of text beginning at the specified line.

(+1) **d** or **Ctrl-D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*:

. or **Ctrl-L**
Causes the current page of text to be redisplayed.

$ Displays the last window full in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(C) are available. They must always be terminated by a newline character, even if the *-n* option is specified.

i/pattern/
Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?
Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The caret (^) notation is useful for terminals which will not properly handle the question mark (?).

After searching, *pg* displays the line found at the top of the screen. You can modify this by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. Use the suffix **t** to restore the original situation.

The following commands modify the environment of perusal:

*i***n** Begins perusing the *i*th next file in the command line. The default value of *i* is 1.

*i***p** Begins perusing the *i*th previous file in the command line. The default value of *i* is 1.

*i***w** Displays another window of text. If *i* is present, set the window size to *i*.

s *filename*
Saves the input in the named file. Only the current file being perused is saved. The white space between the s and *filename* is optional. This command must always be terminated by a newline character, even if the *-n* option is specified.

h Help displays abbreviated summary of available commands.

q or **Q** Quit *pg*.

!command
command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a newline character, even if the *-n* option is specified.

At any time when output is being sent to the terminal, the user can press the quit key (normally Ctrl-\) or the INTERRUPT (BREAK) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, because any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(C), except that a header is printed before each file (if there is more than one).

Example

To use *pg* to read system news, enter:

news | pg -p "(Page %d):"

Files

/etc/termcap	Terminal information data base
/tmp/pg*	Temporary file when input is from a pipe

See Also

ed(C), grep(C), termcap(M)

Notes

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

While waiting for terminal input, *pg* responds to "BREAK and DEL" by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place you in prompt mode. Use these signals with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

The *z* and *f* commands used with *more* are available, and the terminal slash (/), caret (^), or question mark (?) may be omitted from the searching commands.

Standards Conformance

pg is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

pr

prints files on the standard output

Syntax

pr [options] [files]

Description

pr prints the named files on the standard output. If *file* is **-**, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

Options may appear singly or combined in any order. Their meanings are:

+*k* Begins printing with page *k* (default is 1).

-*k* Produces *k*-column output (default is 1). The options **-e** and **-i** are assumed for multicolumn output.

-a Prints multicolumn output across the page.

-m Merges and prints all files simultaneously, one per column (overrides the **-*k***, and **-a** options).

-d Double-spaces the output.

-e*ck* Expands *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every 8th position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any nondigit character) is given, it is treated as the input tab character (default for *c* is the tab character).

-i*ck* In *output*, replaces whitespace wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every 8th position are assumed. If *c* (any nondigit character) is given, it is treated as the output tab character (default for *c* is the tab character).

-n*ck* Provides *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **-m** output. If *c* (any nondigit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

-w*k* Sets the width of a line to *k* character positions (default is 72 for equal-width multicolumn output, no limit otherwise).

-o*k* Offsets each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

-l*k* Sets the length of a page to *k* lines (default is 66).

-h Uses the next argument as the header to be printed instead of the filename.

-p Pauses before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

-f Uses form feed character for new pages (default is to use a sequence of linefeeds). Pauses before beginning the first page if the standard output is associated with a terminal.

-r Prints no diagnostic reports on failure to open files.

-t Prints neither the 5-line identifying header nor the 5-line trailer normally supplied for each page. Quits printing after the last line of each file without spacing to the end of the page.

-s*c* Separates columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

Examples

The following prints *file1* and *file2* as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

The following writes **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr -e9 -t <file1 >file2
```

See Also

cat(C)

Standards Conformance

pr is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ps

reports process status

Syntax

ps [options]

Description

ps prints certain information about active processes. Entering *ps* without any options calls up information about processes associated with the current terminal. The following options control the amount and type of information displayed.

Options

-e Prints information about all processes.

-d Prints information about all processes, except process group leaders.

-a Prints information about all processes, except process group leaders and processes not associated with a terminal.

-f Generates a *full* listing. Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed. Under the **-f** option, *ps* tries to determine and print the process' original command name and arguments. If it cannot, it prints the short listing version of the command name within square brackets. See below for the meaning of columns in a full listing.

-l Generates a *long* listing, including status, priority, location, and memory usage information for each process.

-t *tlist* Restricts listing to data about the processes associated with the terminals given in *tlist*, where *tlist* can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

-p *plist* Restricts listing to data about processes whose process ID numbers are given in *plist*, where *plist* is in the same format as *tlist*.

-u *ulist* Restricts listing to data about processes whose user ID numbers or login names are given in *ulist*, where *ulist* is in the same format as *tlist*. In the listing, the numerical user ID is printed unless the **-f** option is used, in which case the login name is printed.

-g *glist* Restricts listing to data about processes whose process groups are given in *glist*, where *glist* is a list of process group leaders and is in the same format as *tlist*.

-n *namelist* The argument is taken as the name of an alternate *namelist* (**/unix** is the default).

Display Columns

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options only determine what information is provided for a process; they do *not* determine which processes will be listed.

F (l) A status word consisting of flags associated with the process. Each flag is associated with a bit in the status word. These flags are added to form a single octal number. Process flag bits and their meanings are:

01	in core;
02	system process;
04	locked in core (e.g., for physical I/O);
10	being swapped;
20	being traced by another process.

S (l) The state of the process:

0	non-existent;
S	sleeping;
R	running;
I	intermediate;
Z	terminated;
T	stopped;
B	waiting.

UID	(f,l)	The user ID number of the process owner; the login name is printed under the **-f** option. Login names are truncated after 7 characters.
PID	(all)	The process ID; used when killing a process (see *kill*(C)).
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
STIME	(f)	Starting time of the process.
PRI	(l)	The priority of the process; higher numbers mean lower priority.
NI	(l)	Nice value; used in priority computation.
ADDR1, ADDR2	(l)	The memory addresses (physical page frame numbers) of u-area of the process, if resident; otherwise, the disk address. **ADDR1** gives the frame number of the first half of the u-area, and **ADDR2** gives the number of the second half.
SZ	(l)	The size in blocks of the core image of the process, but not including the size of text shared with other processes. Since this size includes the current size of the stack, it will vary as the stack size varies.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
TTY	(all)	The controlling terminal for the process.
TIME	(all)	The cumulative execution time for the process.
CMD	(all)	The command name; the full command name and its arguments are printed under the **-f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Files

/unix	system namelist

/dev/mem	memory
/dev	searched to find swap device and terminal (''tty'') names.

See Also

kill(C), nice(C)

Notes

Things can change while *ps* is running; the picture it gives is only a close approximation to the current system state.

Some data printed for defunct processes are irrelevant.

Authorization

The behavior of this utility is affected by assignment of the mem authorization, which is usually reserved for system administrators. If you do not have this authorization, the output will be restricted to data pertaining to your activities only. Refer to the ''Using a Trusted System'' chapter of the *User's Guide* for more details.

Standards Conformance

ps is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

pscat

ASCII-to-PostScript filter

Syntax

/usr/altos/bin/pscat **[-C]** **[-l** *num*]

Description

The *pscat* utility is a simple filter that converts ASCII text to PostScript®. This utility takes its input from standard input and produces output to standard output. Typically, *pscat* is used in an *lp*(C) printer interface script to prepare ordinary text files for printing on a PostScript printer.

pscat recognizes most PostScript files. So, if a PostScript file is used as input, the text passes through unmodified. Unrecognized files can be forced through with the **-C** option.

The **-l** *num* option allows non-standard paper sizes, where *num* is the number of lines per page. The default is 66.

See Also

lp(C)

Value Added

pscat is an extension of AT&T System V provided by Altos UNIX System V.

pstat

reports system information

Syntax

pstat [**-aipf**] [**-u** ubase1 ubase2] [**-n** namelist] [file]

Description

pstat interprets the contents of certain system tables. *pstat* searches for these tables in **/dev/mem** and **/dev/kmem.** With the *file* given, the tables are sought in the specified *file* rather than **/dev/mem.** The required namelist is taken from **/unix.** Options are:

-a Under **-p**, describe all process slots rather than just active ones.

-i Print the inode table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:
- L Locked
- U Update time *filesystem* (F) must be corrected
- A Access time must be corrected
- M File system is mounted here
- W Wanted by another process (L flag is on)
- T Contains a text file
- C Changed time must be corrected

CNT Number of open file table entries for this inode.

DEV Major and minor device number of file system in which this inode resides.

INO I-number within the device.

MODE Mode bits, see *chmod*(S).

NLK Number of links to this inode.

UID User ID of owner.

SIZ/DEV
Number of bytes in an ordinary file, or major and minor device of special file.

-p Prints process table for active processes with these headings:

LOC The core location of this table entry.

S Run state encoded thus:
- 0 No process
- 1 Waiting for some event
- 3 Runnable

4 Being created
5 Being terminated
6 Stopped under trace

F — Miscellaneous state variables, ORed together:
01 Loaded
02 The scheduler process
04 Locked
010
Swapped out
020
Traced
040
Used in tracing
0100
Locked in by *lock*(S).

PRI — Scheduling priority, see *nice*(S).

SIGNAL — Signals received (signals 1-16 coded in bits 0-15).

UID — Real user ID.

TIM — Time resident in seconds; times over 127 coded as 127.

CPU — Weighted integral of CPU time, for scheduler.

NI — Nice level, see *nice*(S).

PGRP — Process number of root of process group (the opener of the controlling terminal).

PID — The process ID number.

PPID — The process ID of parent process.

ADDR1, ADDR2
If in core, the physical page frame numbers of the u-area of the process. These numbers can be translated into the addresses of the u-area, which is split and stored in two pages. If swapped out, the position in the swap area is measured in multiples of BSIZE bytes.

WCHAN
Wait channel number of a waiting process.

LINK — Link pointer in list of runnable processes.

TEXTP — If text is pure, pointer to location of text table entry (286 only).

INODP — Pointer to location of shared inode (386 only).

CLKT — Countdown for *alarm*(S) measured in seconds.

-u *ubase1 ubase2*
Print information about a user process. *Ubase1 and Ubase2* are the physical page frame numbers of the u-area of the process. The numbers may be obtained by using the long listing (**-l** option) of the *ps*(C) command.

-n *namelist*
Use the file *namelist* as an alternate namelist in place of **/unix.**

-f Print the open file table with these headings:

LOC	The core location of this table entry.
FLG	Miscellaneous state variables: R Open for reading W Open for writing P Pipe
CNT	Number of processes that know this open file.
INO	The location of the inode table entry for this file.
OFFS	The file offset, see *lseek*(S).

Files

/unix Namelist

/dev/mem Default source of tables

See Also

ps(C), stat(S), filesystem(F)

Authorization

The behavior of this utility is affected by assignment of the **mem** authorization, which is usually reserved for system administrators. If you do not have this authorization, the output will be restricted to data pertaining to your activities only. Refer to the ''Using a Trusted System'' chapter of the *User's Guide* for more details.

Value Added

pstat is an extension of AT&T System V provided by Altos UNIX System V.

ptar

process tape archives

Syntax

ptar -c[bfvw] *device block filename...*
ptar -r[bvw] *device block [filename...]*
ptar -t[fv] *device*
ptar -u[bvw] *device block*
ptar -x[flmovw] *device [filename...]*

Description

Tar reads and writes archive files which conform to the **Archive/Interchange File Format** specified in *IEEE Std. 1003.1-1988*.

Options

The following options are available:

-c Creates a new archive; writing begins at the beginning of the archive, instead of after the last file.

-r Writes names files to the end of the archive.

-t Lists the names of all of the files in the archive.

-u Causes named files to be added to the archive if they are not already there, or have been modified since last written into the archive. This implies the **-r** option.

-x Extracts named files from the archive. If a named file matches a directory whose contents had been written onto the archive, that directory is recursively extracted. If a named file in the archive does not exist on the system, the file is create with the same mode as the one in the archive, except that the set-user-id and get-group-id modes are not set unless the user has appropriate privileges.

If the files exist, their modes are not changed except as described above. The owner, group and modification time are restored if possible. If no *filename* argument is given, the entire contents of the archive is extracted. Note that if several files with the same name are in the archive, the last one will overwrite all earlier ones.

-b Causes *ptar* to use the next argument on the command line as the blocking factor for tape records. The default is 1; the maximum is 20. This option should only be used with raw magnetic tape archives. Normally, the block size is determined automatically when reading tapes.

-f Causes *ptar* to use the next argument on the command line as the name of the archive instead of the default, which is usually a tape drive. If - is specified as a filename *ptar* writes to the standard output or reads from the standard input, whichever is appropriate for the options given. Thus, *ptar* can be used as the head or tail of a pipeline.

-l Tells *ptar* to report if it cannot resolve all of the links to the files being archived. If **-l** is not specified, no error messages are written to the standard output. This modifier is only valid with the **-c**, **-r** and **-u** options.

-m Tells *ptar* not to restore the modification times. The modification time of the file will be the time of extraction. This modifier is invalid with th **-t** option.

-o Causes extracted files to take on the user and group identifier of the user running the program rather than those on the archive. This modifier is only valid with the **-x** option.

-v Causes *ptar* to operate verbosely. Usually, *ptar* does its work silently, but the v modifier causes it to print the name of each file it processes, preceded by the option letter. With the **-t** option, v gives more information about the archive entries than just the name.

-w Causes *ptar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no". This modifier is invalid with the **-t** option.

Files

/dev/tty used to prompt the user for information when the **-i** or **-y** options are specified.

See Also

cpio(C), dd(C), find(C), pax(C), pcpio(C)

Copyright

Author

Mark H. Colburn
NAPS International
117 Mackubin Street, Suite 1
St. Paul, MN 55102
mark@jhereg.MN.ORG

Sponsored by **The USENIX Association** for public distribution.

Standards Conformance

ptar is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

purge

overwrites specified files

Syntax

purge [**-f**] [**-r**] [**-v**] [**-m** num] [**-suo**] [**-t** type] ... [**-z**] [files] ...

Description

The command is used to overwrite various parts of the system. It overwrites files specified on the command line, or those listed in a policy file maintained by the system administrator. The policy file defines types of files and devices which are purged as a group. The utility can be used to purge individual files, *divvy*(ADM) divisions, *fdisk*(ADM) partitions, or other devices like magnetic tapes and floppies. An option even exists to zero memory.

The optional flags are outlined below:

-f Do not warn about files which are not present or inaccessible. Attempts to purge a floppy which is inaccessible (e.g., the door is open) will always generate a diagnostic on the system console.

-r Recursively purge directories. Without this flag no action is taken upon directories.

-v Verbose operation, list the name of each file as it is overwritten.

-m *num* Overwrite each *file num* times.

-s Overwrite files and devices designated as "system" in the policy file. (Equivalent to "**-t system**".)

-u Overwrite files and devices designated as "user" in the policy file. (Equivalent to "**-t user**".)

-o Overwrite other (non-system and non-user) files and filesystems. This purges all entries in the policy file which are not of either type **system** or **user**. This flag, by the nature of its implicit definition, has no "**-t**" equivalent.

-t *type* Overwrite the files identified in the policy file as being part of group *type*.

-z Writes binary zeroes to system memory, including memory buffers of intelligent devices (i.e. disk controller cache, etc.). This will close down the system immediately. This should only be done from single-user mode, or when no users are logged on. The system will autoboot if so configured (see *autoboot*(M)). Only the superuser may use this option.

files Regular, directory or special files to purge.

Similarly to regular files, most special files can be purged by being placed in the policy file or with the command **purge /dev/***special_file*. Block special files and some character special files can be overwritten. The console, ttys, printers and other *infinite output* devices cannot be purged with this command. Disks, floppies and magnetic tapes can be overwritten. Tape devices are first erased once and then overwritten the specified number of times.

When both *types* and *files* are specified on the command line, all of the indicated files are overwritten by the utility. In particular, first the files selected from the policy file, and then those specified on the command line, are overwritten.

Each line in the policy file (**/etc/default/purge**) designates a file, filesystem or device as a member of some *type*. The syntax of a line is:

file *type* [*count*]

The optional *count* field is the number of times to overwrite *file*. The default *count* is one. The utility will overwrite *file* any time the command

purge -t *type*

is given.

Blank lines in the policy file and lines beginning with '#' are ignored.

Files

/etc/default/purge The policy file

See Also

autoboot(ADM), dd(C), hd(C), od(C), rm(C), purge(F), sysadmsh(ADM)

Diagnostics

purge: warning: invalid entry in policy file (line *n*)

An invalid line was read from the policy file where *n* is the number of the incorrectly formatted line.

purge: *filename* **is a directory**

If the **-r** switch is not specified no action is taken upon directories and this diagnostic is displayed.

purge: only the superuser can zero memory

This message is displayed when a user other than the superuser tries to use the **-z** option.

Notes

When files are overwritten multiple times, the first pass writes binary zeros. Subsequent passes alternate writing binary ones and binary zeros.

After being overwritten, *od*(C), *dd*(C) or *hd*(C) may be used to verify that no data remains on the device or in the file.

Only the superuser may use the **-z** option to zero the system's memory.

Value Added

purge is an extension of AT&T System V provided by Altos UNIX System V.

pwcheck

checks password file

Syntax

pwcheck [file]

Description

pwcheck scans the password file and checks for any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is **/etc/passwd.**

Files

/etc/passwd

See Also

grpcheck(C), group(F), passwd(F)

pwd

prints working directory name

Syntax

pwd

Description

pwd prints the pathname of the working (current) directory.

See Also

cd(C)

Diagnostics

"Cannot open .." and "Read error in .." indicate possible file system trouble. In such cases, see the *System Administrator's Guide* for information on fixing the filesystem.

Standards Conformance

pwd is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

quot

summarizes file system ownership

Syntax

quot [option] ... [filesystem]

Description

quot prints the number of blocks in the named *filesystem* currently owned by each user. If no *filesystem* is named, the file systems given in **/etc/mnttab** are examined.

The following options are available:

-n Processes standard input. This option makes it possible to produce a list of all files and their owners with the following command:

ncheck filesystem | sort +0n | quot -n filesystem

-c Prints three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file. Data for files of size greater than 499 blocks are included in the figures for files of exactly size 499.

-f Prints a count of the number of files as well as space owned by each user.

Files

/etc/passwd	Gets user names
/etc/mnttab	Contains list of mounted file systems

See Also

cmchk(C), du(C), ls(C), machine(HW)

Notes

Holes in files are counted as if they actually occupied space.

Blocks are reported in 512 byte blocks.

See also *Notes* under *mount*(ADM).

random

generates a random number

Syntax

random [-s] [scale]

Description

random generates a random number on the standard output. and returns the number as its exit value. By default, this number is either 0 or 1 (i.e., *scale* is 1 by default). If *scale* is given a value between 1 and 255, then the range of the random value is from 0 to *scale*. If *scale* is greater than 255, an error message is printed.

When the **-s**, ''silent'' option is given, the random number is returned as an exit value but is not printed on the standard output. If an error occurs, *random* returns an exit value of zero.

See Also

rand(S)

Notes

This command does not perform any floating point computations.

random uses the time of day as a seed.

rcp

copies files across systems

Syntax

rcp [options] [srcmachine:]srcfile [destmachine:]destfile

Description

rcp copies files between systems in a Micnet network. The command copies the *srcmachine:srcfile* to *destmachine:destfile* , where *srcmachine*: and *destmachine*: are optional names of systems in the network, and *srcfile* and *destfile* are pathnames of files. If a machine name is not given, the name of the current system is assumed. If - is given in place of *srcfile*, *rcp* uses the standard input as the source. Directories named on the destination machine must have write permission, and directories and files named on a remote source machine must have read permission.

The available options are:

-m
Mails and reports completion of the command, whether there is an error or not.

-u [*machine*:]*user*
Any mail goes to the named *user* on *machine*. The default *machine* is the machine on which the *rcp* command is completed or on which an error was detected. If an alias for *user* exists in the system alias files on that *machine*, the mail will be redirected to the appropriate mailbox(es). Since system alias files are usually identical throughout the network, any specified *machine* will most likely be overridden by the aliasing mechanism. To prevent aliasing, *user* must be escaped with at least two \ characters (at least four if given as a shell command).

rcp is useful for transferring small numbers of files across the network. The network consists of daemons that periodically awaken and send files from one system to another. The network must be installed using *netutil* (ADM) before *rcp* can be used.

Also, to enable transfer of files from a remote system, either:

This line should be in */etc/default/micnet* on the systems in the network:

```
rcp=/usr/bin/rcp
```

Or, these lines should be in that file:

```
executeall
execpath=PATH=path
```

where *path* must contain */usr/bin*.

Example

rcp -m machine1:/etc/mnttab /tmp/vtape

See Also

mail(C), micnet(F), netutil(ADM), remote(C)

Diagnostics

If an error occurs, mail is sent to the user.

Notes

Full pathnames must be specified for remote files.

rcp handles binary data files transparently, no extra options or protocols are needed to handle them. Wildcards are not expanded on the remote machine.

rcvtrip

notifies mail sender that recipient is away

Syntax

/usr/bin/rcvtrip [-d] [address]

Description

The *rcvtrip* command makes it possible for you to notify the sender of a message that you are on holiday and you won't be answering your mail for some time. MMDF runs *rcvtrip* on your behalf rather than by you directly.

To enable use of *rcvtrip*, put the following line in your **.maildelivery** file:

```
*     -     pipe     R     rcvtrip $(sender)
```

Make sure that your **.maildelivery** file is not writable by anyone but you. You may also place a ''custom'' reply message in a file named **tripnote**. Finally, you should create an empty **triplog** file.

When *rcvtrip* processes a message, it performs the following steps:

1. Decide if this type of message should receive a reply.

2. Decide to whom the reply should be sent.

3. Decide whether this sender has already gotten a reply.

The *rcvtrip* command decides whether this is the type of message that should get a reply by looking at the contents of the "Resent-To:", "Resent-Cc:", "To:" and "Cc:" header fields. If the recipient has an **.alter_egos** file (described next), then one of the addresses in that file must appear in the one of these header fields for a reply to be sent. If the recipient does not have an **.alter_egos** file, then the recipient's name or a first-order alias of the recipient's name (for example, dlong-->long) must appear in one of these header fields for a reply to be sent. This procedure ensures that *rcvtrip* will not reply to messages sent to mailing lists, unless the recipient's name (or some variant of the recipient's name) is explicitly mentioned in a header field.

If *rcvtrip* has decided that it should send a reply for the message, then it looks at several other address fields to determine to whom the reply should be sent. It uses, in order of precedence:

1. addresses in 'Resent-Reply-To:'

2. addresses in 'Resent-From:' and, if present, 'Resent-Sender:'

3. addresses in 'Reply-To:'

4. addresses in 'From:' and either 'Sender:', if present, or the **address** argument from the command line.

The *rcvtrip* command notifies any originator of mail who has not previously been notified unless you pre-load their address into the **triplog** file (refer to the Files section). The reply begins with some standard text (supplied by *rcvtrip*) followed by whatever text the user has placed in the **tripnote** file (or a canned message if the **tripnote** file is missing). The originators' names are recorded in **triplog**, along with the date and time the message came in, an indication of whether it was answered ('+'=yes), and the first few characters of the subject. This appears as:

```
+ jpo@nott.ac.uk    Wed Oct 8 16:08 >> about your last message
```

Files

$HOME/tripnote contains a reply message to be sent to those sending you mail.

$HOME/triplog contains a list of who sent a message, what was its subject, when it arrived, and if a response was sent. It can also be initialized by hand to contain the addresses, one per line, which are not to receive replies.

$HOME/logfile, it it exists, becomes an output file for logging diagnostic information. If the **-d** option is specified, then extensive output is generated for debugging purposes. It is not a good idea to leave **-d** enabled if this file is left lying around, as the output can be quite voluminous.

$HOME/.alter_egos, an optional file composed of 'user@domain' lines for all addresses to be considered 'you'. This is needed if you have multiple hosts forwarding their mail to you. If this file is present, then the standard comparisons against your username and first-level aliases of your username do not occur.

$HOME/.maildelivery is your mail delivery specification file. The previous example shows the line that should be added to **.maildelivery** to enable use of *rcvtrip*. In this line, the $(sender) argument is optional (but recommended). You may need to give the full pathname of *rcvtrip* if it is not in your search path.

See Also

maildelivery(F)

remote

executes commands on a remote system

Syntax

remote [-] [-f file] [-m] [-u user] machine
command [arguments]

Description

remote is a limited networking facility that permits execution of Altos UNIX System V commands across serial lines. Commands on any connected system may be executed from the host system using *remote*. A command line consisting of *command* and any blank-separated *arguments* is executed on the remote *machine*. A machine's name is located in the file **/etc/systemid**. Note that wild cards are *not* expanded on the remote machine, so they should not be specified in *arguments*. The optional **-m** switch causes mail to be sent to the user telling whether the command is successful.

The available options follow:

- A dash signifies that standard input is used as the standard input for *command* on the remote *machine*. Standard input comes from the local host and not from the remote machine.

-f *file* Use the specified *file* as the standard input for *command* on the remote *machine*. The *file* exists on the local host and not on the remote machine.

-m Mails the user to report completion of the command. By default, mail reports only errors.

-u *user* Any mail goes to the named *user* on *machine*. The default *machine* is the machine on which an error was detected, or on which the *remote* command was completed. The mail will be redirected to the appropriate mailbox(es), if an alias for *user* exists in the system alias files on that *machine* . Since system alias files are usually identical throughout the network, any specified *machine* will most likely be overridden by the aliasing mechanism. To prevent aliasing, *user* must be escaped with at least two \ characters (at least four if given as a shell command).

Before *remote* can be successfully used, a network of systems must first be set up and the proper daemons initialized using *netutil* (ADM). Also, entries for the command to be executed using *remote* must be added to the */etc/default/micnet* files on each remote machine.

Example

The following command executes an *ls* command on the directory **/tmp** of the machine *machine1*:

remote machine1 ls /tmp

See Also

rcp(C), mail(C), netutil(ADM), micnet(F)

Notes

The *mail* command uses the equivalent of *remote* to send mail between machines.

rm

removes files or directories

Syntax

rm [**-fri**] file ...

Description

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If the user does not have write permission on a specified file and the standard input is a terminal, the user is prompted for confirmation. The file's name and permissions are printed and a line is read from the standard input. If that line begins with y, the file is deleted, otherwise the file remains. If the **-f** option is given or if the standard input is not a terminal, no messages are issued; files are simply removed.

rm will not delete directories unless the **-r** option is used.

Options

The following options are recognized.

-f When invoked with the **-f** option, *rm* does not prompt the user for confirmation for files on which the user does not have write permission. The files are simply removed.

-r The **-r** (recursive) option causes *rm* to recursively delete the entire contents of the any directories specified, and the directories themselves. Note that the *rmdir*(C) command is a safer way of removing directories.

-i The **-i** (interactive) option causes *rm* to ask whether to delete each file, and if the **-r** option is in effect, whether to examine each directory.

The special option "--" can be used to delimit options. For example, a file named "-f" could not be removed by *rm* because the hyphen is interpreted as an option; the command **rm -f** would do nothing, since no file is specified. Using **rm -- -f** removes the file successfully.

See Also

rmdir(C)

Notes

It is forbidden to remove the file .. to avoid the consequences of inadvertently doing something like:

```
rm -r .*
```

It is also forbidden to remove the root directory of a given file system.

No more than 17 levels of subdirectories can be removed using the **-r** option.

Standards Conformance

rm is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

rmdir

removes directories

Syntax

rmdir [**-p**] [**-s**] dirname ...

Description

rmdir removes the entries for one or more subdirectories from a directory. A directory must be empty before it can be removed. Note that the "rm -r *dir*" (command is a more dangerous alternative to *rmdir.)* If the parent directory has the sticky bit set, removal occurs only if one of the following is true:

the parent directory is owned by the user
the *dirname* directory is owned by the user
the *dirname* directory is writable to the user
the user is the super-user

The **-p** option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

The **-s** option is used to suppress the message printed on standard error when **-p** is in effect.

rmdir will refuse to remove the root directory of a mounted filesystem.

See Also

rm(C)

Diagnostics

rmdir returns an exit code of 0 if all the specified directories are removed successfully. Otherwise, it returns a non-zero exit code.

Standards Conformance

rmdir is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

rsh

invokes a restricted shell (command interpreter)

Syntax

rsh [flags] [name [arg1 ...]]

Description

rsh is a restricted version of the standard command interpreter *sh*(C). It is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that changing directory with *cd*, setting the value of $PATH, using command names containing slashes, and redirecting output using > and >> are all disallowed.

When invoked with the name **-rsh**, *rsh* reads the user's **.profile** (from **$HOME/.profile**). It acts as the standard *sh* while doing this, except that an interrupt causes an immediate exit, instead of causing a return to command level. The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end user shell procedures that have access to the full power of the standard shell, while restricting him to a limited menu of commands; this scheme assumes that the end user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions, then leaving the user in an appropriate directory (probably *not* the login directory).

rsh is actually just a link to *sh* and any *flags* arguments are the same as for *sh*(C).

The system administrator often sets up a directory of commands that can be safely invoked by *rsh*.

See Also

sh(C), profile(M)

sddate

prints and sets backup dates

Syntax

sddate [name lev date]

Description

If no argument is given, the contents of the backup date file **/etc/ddate** are printed. The backup date file is maintained by *backup*(C) and contains the date of the most recent backup for each backup level for each filesystem.

If arguments are given, an entry is replaced or made in **/etc/ddate.** *name* is the last component of the device pathname, *lev* is the backup level number (from 0 to 9), and *date* is a time in the form taken by *date*(C):

mmddhhmm[yy]

Where the first *mm* is a two-digit month in the range 01-12, *dd* is a two-digit day of the month, *hh* is a two-digit military hour from 00-23, and the final *mm* is a two-digit minute from 00-59. An optional two-digit year, *yy,* is presumed to be an offset from the year 1900, i.e., 19*yy*.

Some sites may wish to back up file systems by copying them verbatim to backup media. *sddate* could be used to make a "level 0" entry in **/etc/ddate**, which would then allow incremental backups.

For example:

sddate rhda 5 10081520

makes an **/etc/ddate** entry showing a level 5 backup of **/dev/rhda** on October 8, at 3:20 PM.

Files

/etc/ddate

See Also

backup(C), dump(C), date(C)

Diagnostics

bad conversion If the date set is syntactically incorrect.

sdiff

compares files side-by-side

Syntax

sdiff [options ...] file1 file2

Description

sdiff uses the output of *diff*(C) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1* , a > in the gutter if the line only exists in *file2* , and a | for lines that are different.

For example:

```
x    |    y
a         a
b    <
c    <
d         d
     >    c
```

The following options exist:

-w *n* Uses the next argument, *n*, as the width of the output line. The default line length is 130 characters.

-l Only prints the left side of any lines that are identical.

-s Does not print identical lines.

-o *output* Uses the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2* . Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(C), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l Appends the left column to the output file

r Appends the right column to the output file

s Turns on silent mode; does not print identical lines

v Turns off silent mode

e l
Calls the editor with the left column

e r
Calls the editor with the right column

e b
Calls the editor with the concatenation of left and right

e Calls the editor with a zero length file

q Exits from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

See Also

diff(C), ed(C)

sed

invokes the stream editor

Syntax

sed [**-n**] [**-e** script] [**-f** sfile] [files]

Description

sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **-f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **-e** option and no **-f** options, the flag **-e** may be omitted. The **-n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a D command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

A semicolon (;) can be used as a command delimiter.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed*(C) modified as follows:

- In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.
- The escape sequence **\n** matches a newline *embedded* in the pattern space.
- A period **.** matches any character except the *terminal* newline of the pattern space.

- A command line with no addresses selects every pattern space.
- A command line with one address selects each pattern space that matches the address.
- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter, the process is repeated, looking again for the first address.

Editing commands can be applied only to nonselected pattern spaces by use of the negation function **!** (below).

In the following list of functions, the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with backslashes to hide the newlines. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**
text Appends *text*, placing it on the output before reading the next input line.

(2) **b** *label* Branches to the **:** command bearing the *label*. If *label* is empty, branches to the end of the script.

(2) **c**
text Changes text by deleting the pattern space and then appending *text*. With 0 or 1 address or at the end of a 2-address range, places *text* on the output and starts the next cycle.

(2) **d** Deletes the pattern space and starts the next cycle.

(2) **D** Deletes the initial segment of the pattern space through the first newline and starts the next cycle.

(2) **g** Replaces the contents of the pattern space with the contents of the hold space.

(2) **G** Appends the contents of the hold space to the pattern space.

(2) **h** Replaces the contents of the hold space with the contents of the pattern space.

(2) **H** Appends the contents of the pattern space to the hold space.

(1) **i**
text Insert. Places *text* on the standard output.

(2) **l** Lists the pattern space on the standard output with non-printing characters spelled in two-digit ASCII and long lines folded.

(2) **n** Copies the pattern space to the standard output. Replaces the pattern space with the next line of input.

(2) **N** Appends the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) **p** Prints (copies) the pattern space on the standard output.

(2) **P** Prints (copies) the initial segment of the pattern space through the first newline to the standard output.

(1) **q** Quits *sed* by branching to the end of the script. No new cycle is started.

(2) **r** *rfile* Reads the contents of *rfile* and places them on the output before reading the next input line.

(2)s/*regular expression*/*replacement*/*flags*

Substitutes the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a more detailed description, see *ed*(C). *Flags* is zero or more of:

n n=1-512. Substitute for just the nth occurrence of the *regular expression.*

g Globally substitutes for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Prints the pattern space if a replacement was made.

w *wfile*
Writes the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Branches to the colon (:) command bearing *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t** command. If *label* is empty, **t** branches to the end of the script.

(2) **w** *wfile* Writes the pattern space to *wfile*.

(2) **x** Exchanges the contents of the pattern and hold spaces.

(2) **y**/*string1* /*string2* /
Replaces all occurrences of characters in *string1* with the corresponding characters in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function*
Applies the *function* (or group, if *function* is { }) only to lines *not* selected by the address(es).

(0) **:** *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) **=** Places the current line number on the standard output as a line.

(2) **{** Executes the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

See Also

awk(C), ed(C), grep(C)

Notes

This command is explained in detail in the *User's Guide*.

Standards Conformance

sed is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

setcolor, setcolour

set screen color and other screen attributes

Syntax

setcolor -[nbrgopc] argument [argument]

Description

setcolor allows the user to set the screen color on a color screen. Both foreground and background colors can be set independently in a range of 16 colors. **setcolor** can also set the reverse video and graphics character colors. **setcolor** with no arguments produces a usage message that displays all available colors, then resets the screen to its previous state.

For example, the following strings are possible colors.

blue	magenta	brown	black
lt_blue	lt_magenta	yellow	gray
cyan	white	green	red
lt_cyan	hi_white	lt_green	lt_red

The following flags are available. In the arguments below, ''color'' is taken from the above list.

-n Set the screen to ''normal'' white characters on black background.

color [*color*]
Set the foreground to the first color. Sets background to second color if a second color choice is specified.

-b *color*
Set the background to the specified color.

-r *color color*
Set the foreground reverse video characters to the first color. Set reverse video characters' background to second color.

-g *color color*
Set the foreground graphics characters to the first color. Set graphics characters' background to second color.

-o Set the color of the screen border (overscan region).

-p *pitch duration*
Set the pitch and duration of the bell. Pitch is the period in microseconds, and duration is measured in fifths of a second. When using this option, a control-G (bell) must be echoed to the screen for the command to work. For example:

```
setcolor -p 2500 2
echo ^G
```

-c *first last*
Set the first and last scan lines of the cursor. (For more information see *screen*(HW).)

Notes

The ability of *setcolor* to set any of these described functions is ultimately dependent on the ability of devices to support them. *setcolor* emits an escape sequence that may or may not have an effect on monochrome devices.

Occasionally changing the screen color can help prolong the life of your monitor.

See Also

screen(HW)

Value Added

setcolor and *setcolour* are extensions of AT&T System V provided by Altos UNIX System V.

setkey

assigns the function keys

Syntax

setkey *keynum string*

Description

The **setkey** command assigns the given ANSI *string* to be the output of the computer function key given by *keynum*. For example, the command:

setkey 1 date

assigns the string "date" as the output of function key 1. The *string* can contain control characters, such as a newline character, and should be quoted to protect it from processing by the shell. For example, the command:

setkey 2 "pwd ; lc\n"

assigns the command sequence "pwd ; lc" to function key 2. Notice how the newline character is embedded in the quoted string. This causes the commands to be carried out when function key 2 is pressed. Otherwise, the Enter key would have to be pressed after pressing the function key, as in the previous example.

setkey translates ^ into ^^, which, when passed to the screen driver, is interpreted as a right angle bracket (>), or greater than key.

Notes

setkey works only on the *console* keyboard.

The string mapping table is where the function keys are defined. It is an array of 512 bytes (typedef *strmap_t*) where null terminated strings can be put to redefine the function keys. The first null terminated string is assigned to the first string key, the second to the second string key, and so on. There is one string mapping table per multiscreen.

Although the size of the **setkey** string mapping table is 512 bytes, there is a limit of 30 characters that can be assigned to any individual function key.

Assigning more than 512 characters to the string mapping table causes the function key buffer to overflow. When this happens, the sequences sent by the arrow keys are overwritten, effectively disabling them. Once the function key buffer overflows, the only way to enable the arrow keys is to reboot the system.

The table below lists the *keynum* values for the function keys:

Function key	*keynum*	**Function key**	*keynum*
F1	1	Ctrl-F10	34
F2	2	Ctrl-F11	35
F3	3	Ctrl-F12	36
F4	4	Ctrl-Shift-F1	37
F5	5	Ctrl-Shift-F2	38
F6	6	Ctrl-Shift-F3	39
F7	7	Ctrl-Shift-F4	40
F8	8	Ctrl-Shift-F5	41
F9	9	Ctrl-Shift-F6	42
F10	10	Ctrl-Shift-F7	43
F11	11	Ctrl-Shift-F8	44
F12	12	Ctrl-Shift-F9	45
Shift-F1	13	Ctrl-Shift-F10	46
Shift-F2	14	Ctrl-Shift-F11	47
Shift-F3	15	Ctrl-Shift-F12	48
Shift-F4	16		
Shift-F5	17	**Numeric Key-Pad**	*keynum*
Shift-F6	18		
Shift-F7	19	7	49
Shift-F8	20	8	50
Shift-F9	21	9	51
Shift-F10	22	-	52
Shift-F11	23	4	53
Shift-F12	24	5	54
Ctrl-F1	25	6	55
Ctrl-F2	26	+	56
Ctrl-F3	27	1	57
Ctrl-F4	28	2	58
Ctrl-F5	29	3	59
Ctrl-F6	30	0	60
Ctrl-F7	31		
Ctrl-F8	32		
Ctrl-F9	33		

For a table of the escape sequences, refer to keyboard(HW).

Files

/bin/setkey

See Also

keyboard(HW)

setmode

Port modes utility

Syntax

setmode *device mode ...*

Description

The *setmode* utility sets tty modes (see *tty*(M)) for tty ports that are being used for serial devices. (Use this utility to set baud rate, tab expansion, and newline actions for programs that communicate directly through a serial port.)

This utility takes a list of tty *modes* from its command line, performs *stty*(C) on the indicated *device*, and sleeps forever, which keeps *device* open with the desired modes. Invoke *setmode* once for each port device.

To ensure that *setmode* is run every time the system enters multiuser mode, invoke *setmode* in the **/etc/inittab** file.

You must invoke *setmode* with at least two arguments: the name of the *device* special file (for example, /dev/tty01), and at least one tty *mode*.

Files

/dev/tty*	tty devices
/etc/inittab	

Related Commands

disable(C), enable(C), stty(C), inittab(F), pcu(ADM),

See Also

tty(M)

Value Added

setmode is an extension of AT&T System V provided by Altos UNIX System V.

sh

invokes the shell command interpreter

Syntax

sh [**-aceiknrstuvx**] [args]

Description

The shell is the standard command programming language that executes commands read from a terminal or a file. See *Invocation* below for the meaning of arguments to the shell.

Commands

A *simple-command* is a sequence of nonblank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(S)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 1000+*status* if it terminates abnormally (i.e., if the failure produces a core file). See *signal*(S) for a list of status values.

A *pipeline* is a sequence of one or more *commands* separated by a vertical bar (|). (The caret (^), is an obsolete synonym for the vertical bar and should not be used in a pipeline.) The standard output of each command but the last is connected by a *pipe*(S) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (nonzero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following commands. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command:

for *name* [**in** *word* ...]
do
list
done

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in**
[*pattern* [| *pattern*] ...) *list*
;;]
esac

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see *Filename Generation* below).

if *list* **then**
list
[**elif** *list* **then**
list]
[**else** *list*]
fi

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list*
do
list
done

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)
Executes *list* in a subshell.

{*list*;}
list is simply executed.

name () {*list;*}

Define a function which is referenced by *name*. The body of functions is the *list* of commands between { and }. Execution of functions is described later (see *Execution.*)

The following words are recognized only as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (` `) may be used as part or all of a word; trailing newlines are removed.

No interpretation is done on the command string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape grave accents (`) or other backslashes and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies withing a pair of double quotes (" ' ... ' "), backslashes used to escape a double quote (\") will be removed; otherwise, they will be left intact.

If a backslash is used to escape a newline character, both the backslash and the newline are removed (see the section on "Quoting"). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has not effect. Backslashes that precede characters other than \, `, ", **newline,** and $ are left intact.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by set. Keyword parameters, (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same name.

${*parameter*}
A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *, @, #, ?, -, $, and !. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is * or @, then all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:-*word*}
If *parameter* is set and is not a null argument, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null, then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is not a null argument, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}
If *parameter* is set and is not a null argument, substitute *word*; otherwise substitute nothing. In the above, *word* is not evaluated unless it is to be used as the substituted string, so that in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal

- Flags supplied to the shell on invocation or by the **set** command

? The decimal value returned by the last synchronously executed command

$ The process number of this shell

! The process number of the last background command invoked

The following parameters are used by the shell:

CDPATH
Defines search path for the *cd* command. See the section *Special Commands*, "cd".

HOME
The default argument (home directory) for the *cd* command

PATH
The search path for commands (see *Execution* below)

MAIL
If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by **%** and a message that will be printed when the modification time changes. The default message is *you have mail.*

PS1
Primary prompt string, by default "$ "

PS2
Secondary prompt string, by default "> "

IFS
Internal field separators, normally **space**, **tab**, and **newline**

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(ADM) and *accton*(ADM) can be used to analyze the data collected. This feature does not work with all versions of the shell.

SHELL

When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH, PS1, PS2,** and **IFS,** while **HOME** and **MAIL** are not set at all by the shell (although **HOME** *is* set by *login*(M)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character . at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly. These characters and their matching patterns are:

* Matches any string, including the null string.

? Matches any single character.

[...]
Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening bracket ([) is an exclamation mark (!), then any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > newline space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **newline** is ignored. All characters enclosed between a pair of single quotation marks (´´), except a single quotation mark, are quoted. Inside double quotation marks (""), parameter and command substitution occurs and \ quotes the characters \, `, ", and $. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2"** ...

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Spelling Checker

When using *cd*(C) the shell checks spelling. For example, if you change to a different directory using *cd* and misspell the directory name, the shell responds with an alternative spelling of an existing directory. Enter "y" and press RETURN (or just press RETURN) to change to the offered directory. If the offered spelling is incorrect, enter "n", then retype the command line. In this example the **sh**(C) response is boldfaced:

$ cd /usr/spol/uucp
cd /usr/spool/uucp?y
ok

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command*. They are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word* — Use file *word* as standard input (file descriptor 0).

>*word* — Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.

>*word* — Use file *word* as standard output. If the file exists, output is appended to it (by first seeking the end-of-file); otherwise, the file is created.

<[-]*word* — The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\newline** is ignored, and \ must be used to quote the characters

\, **$**, **`**, and the first character of *word*. If **-** is appended to ®all leading tabs are stripped from *word* and from the document.

<&digit The standard input is duplicated from file descriptor *digit* (see *dup*(S)). Similarly for the standard output using **>**.

<&- The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see *environ*(M)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affect the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 cmd args

and

(export TERM; TERM=450; cmd args)

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name.

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11. See the **trap** command below.

Execution

Each time a command is executed, the above substitutions are carried out. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1, $2,** ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec* (S).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a /, then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Shell procedures are often used by users running the **csh**. However, if the first character of the procedure is a # (comment character), **csh** assumes the procedure is a **csh** script, and invokes */bin/csh* to execute it. Always start **sh** procedures with some other character if **csh** users are to run the procedure at any time. This invokes the standard shell */bin/sh*.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see **hash** in next section).

Special Commands

Input/output redirection is permitted for these commands:

: No effect; the command does nothing. A zero exit code is returned.

. *file*
Reads and executes commands from *file* and returns. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exits from the enclosing **for** or while loop, if any. If *n* is specified, it breaks *n* levels.

continue [*n*]
Resumes the next iteration of the enclosing **for** or **while** loop. If *n* is specified, it resumes at the *n*-th enclosing loop.

cd [*arg*]
Changes the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, the search path is not used. Otherwise, each directory in the path is searched for *arg*.

If the shell is reading its commands from a terminal, and the specified directory does not exist (or some component cannot be searched), spelling correction is applied to each component of *directory*, in a search for the "correct" name. The shell then asks whether or not to try and change directory to the corrected directory name; an answer of *n* means "no", and anything else is taken as "yes".

echo [arg]
Writes arguments separated by blanks and terminated by a newline on the standard output. Arguments may be enclosed in quotes. Quotes are required so that the shell correctly interprets these special escape sequences:

\b Backspace

\c Prints line without newline.

\f Form feed

\n Newline

\r Carriage return

\t Tab

\v Vertical tab

\\ Backslash

n The 8-bit character whose ASCII code is the 1, 2 or 3-digit octal number *n* must start with a zero

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end-of-file will also cause the shell to exit.

export [*name* ...]
The given *name*s are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, a list of all names that are exported in this shell is printed.

getopts
Use in shell scripts to support command syntax standards (see *intro*(C)); it parses positional parameters and checks for legal options. See *getopts*(C) for usage and description.

hash [**-r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* ...

pwd
Print the current working directory. See *pwd*(C) for usage and description.

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**-eknuvx** [*arg* ...]]

- -a Mark variables which are modified or created for export.
- **-e** If the shell is noninteractive, exits immediately if a command exits with a nonzero exit status.
- **-f** Disables file name generation.
- **-h** Locates and remembers function commands as functions are defined (function commands are normally located when the function is executed).
- **-k** Places all keyword arguments in the environment for a command, not just those that precede the command name.
- **-n** Reads commands but does not execute them.
- **-u** Treats unset variables as an error when substituting.
- **-v** Prints shell input lines as they are read.
- **-x** Prints commands and their arguments as they are executed. Although this flag is passed to subshells, it does not enable tracing in those subshells.
- **--** Does not change any of the flags; useful in setting $1 to -.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$-**. The remaining arguments are positional parameters and are assigned, in order, to **$1, $2, ...** If no arguments are given, the values of all names are printed.

shift
The positional parameters from **$2** ... are renamed **$1** ...

test
Evaluates conditional expressions. See *test*(C) for usage and description.

times
Prints the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
arg is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. The highest signal number allowed is 16. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is

ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]
imposes a size limit of *n* blocks on files written by the shall and its child processes (files of any size may be read). Any user may decrease the file size limit, but only the super-user (root) can increase the limit. With no argument, the current limit is printed.

If no option is given and a number is specified, **-f** is assumed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

umask [*ooo*]
The user file-creation mask is set to the octal number *ooo* where *o* is an octal digit (see *umask*(C)). If *ooo* is omitted, the current value of the mask is printed.

wait [*n*]
Waits for the specified process to terminate, and reports the termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is always 0.

Invocation

If the shell is invoked through *exec*(S) and the first character of argument 0 is -, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c *string* If the **-c** flag is present, commands are read from *string*.

-s If the **-s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

-t If the **-t** flag is present, a single command is read and executed, and the shell exits. This flag is intended for use by C programs only and is not useful interactively.

-i If the **-i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case, TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

-r If the **-r** flag is present, the shell is a restricted shell (see *rsh*(C)).

The remaining flags and arguments are described under the **set** command above.

Exit Status

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed. See the **exit** command above.

Files

/etc/profile	system default *profile* if none is present
$HOME/.profile	read by login shell at login
/tmp/sh*	temporary file for <<
/dev/null	source of empty file

See Also

cd(C), env(C), login(M), newgrp(C), rsh(C), test(C), umask(C), dup(S), exec(S), fork(S), pipe(S), signal(S), umask(S), wait(S), a.out(F), profile(M), environ(M)

Notes

The command **readonly** (without arguments) produces the same output as the command **export**.

If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

When a *sh*(C) user logs in, the system reads and executes commands in */etc/profile* before executing commands in the user's *$HOME/.profile*. You can, therefore, modify the environment for all *sh*(C) users on the system by editing */etc/profile*.

The shell doesn't treat the high (eighth) bit in the characters of a command line argument specially, nor does it strip the eighth bit from the characters of error messages. Previous versions of the shell used the eighth bit as a quoting mechanism.

Existing programs that set the eighth bit of characters in order to quote them as part of the shell command line should be changed to use of the standard shell quoting mechanisms (see the section on "Quoting").

Words used to filenames in input/output redirection are not interpreted for filename generation (see the section on "File Name Generation"). For example, **cat file1 > a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message:

    fork failed - too many processes

try using the *wait*(C) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes (there is a limit to the number of processes that can associated with your login, and to the number the system can keep track of.).

Warnings

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

Not all processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For **wait** *n*, if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

Standards Conformance

sh is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

shl

shell layer manager

Syntax

shl

Description

shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer that can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty*(C) option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *shl* has its own prompt, **>**, to help distinguish it from a layer.

A *layer* is a shell that has been bound to a virtual tty device (**/dev/sxt???**). The virtual device can be manipulated like a real tty device using *stty*(C) and *ioctl*(S). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or newline. Only the first eight characters are significant. The *names* **(1)** through **(7)** cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create [*name*]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer.

The shell prompt variable **PS1** is set to the name of the layer followed by a space, or, if superuser, the name followed by a sharp (#) and a space. A maximum of seven layers can be created.

block *name* [*name* ...]
For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

delete *name* [*name* ...]
For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)).

help (or ?)
Print the syntax of the *shl* commands.

layers [**-l**] [*name* ...]
For each *name*, list the layer name and its process group. The **-l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle
Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

quit
Exit *shl*. All layers are sent the SIGHUP signal.

name
Make the layer referenced by *name* the current layer.

Files

/dev/sxt???	Virtual tty devices
$SHELL	Variable containing path name of the shell to use (default is /bin/sh).

See Also

ioctl(S), mkdev(ADM), sh(C), signal(S), stty(C), sxt(M)

Note

It is inadvisable to kill *shl.*

If *shl* does not run properly on a particular terminal, you may have to set **istrip** for that terminal's line by entering the following command at the terminal:

stty istrip

By default, the Operating System is not configured for shell layers. To add this to kernel, use the command:

mkdev shl

This executes a script which prompts you for the number of sessions desired. The script also allows you to relink the kernel. The new session limit becomes effective after the kernel is rebooted. (For more information, see *mkdev*(ADM).)

Standards Conformance

shl is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

sleep

suspends execution for an interval

Syntax

sleep time

Description

sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

See Also

alarm(S), sleep(S)

Notes

It is recommended that *time* be less than 65536 seconds. If this amount is exceeded, *time* will be arbitrarily set to some value less than 65536 seconds.

Standards Conformance

sleep is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

sort

sorts and merges files

Syntax

sort [-cmu] [-ooutput] [-ykmem] [-zrecsz] **[-dfiMnr]** [-**btx**] [+pos1] [-pos2] [files]

Description

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if - is used as a file name or if no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is determined by the collating sequence defined by the locale (see *locale*(M)).

The following options alter the default behavior:

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merge only, the input files are already sorted.

-u Unique: suppress all but one in each set of lines having equal keys. This option can result in unwanted characters placed at the end of the sorted file.

-o*output*
The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **-o** and *output*.

-y*kmem*
The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, -y0 is guaranteed to start with minimum memory. By convention, -y (with no argument) starts with maximum memory.

-zrecsz
Causes *sort* to use a buffer size of *recsz* bytes for the merge phase. Input lines longer than the buffer size will cause *sort* to terminate abnormally. Normally, the size of the longest line read during the sort phase is recorded and this maximum is used as the record size during the merge phase, eliminating the need for the **-z** option. However, when the sort phase is omitted (**-c** or **-m** options) a system default buffer size is used, and if this is not large enough, the **-z** option should be used to prevent abnormal termination.

The following options override the default ordering rules.

-d "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons. Dictionary order is defined by the locale setting (see *locale* (M)).

-f Fold lower case letters into upper case. Conversion between lowercase and uppercase letters are governed by the locale setting (see *locale* (M)).

-i Ignore non-printable characters in non-numeric comparisons. Non-printable characters are defined by the locale setting (see *locale* (M)).

-M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **-M** option implies the **-b** option (see below).

-n An initial numeric string, consisting of optional blanks, an optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field (a minimal sequence of characters followed by a field separator or a newline). By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

-t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

-b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **b**, **d**, **f**, **i**, **n**, or **r**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect, *n* is counted from the first non-blank in the *m*+1st field; *+m*.0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *-m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect, *n* is counted from the last leading blank in the *m*+1st field; *-m*.1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

Examples

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

sort -r +1.0b -1.1b infile1 infile2

Print the password file (*passwd*(F)) sorted by the numeric user ID (the third colon-separated field):

sort -t: +2n -3 /etc/passwd

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

sort -um +2 -3 infile

Files

/usr/tmp/stm???

See Also

coltbl(M), comm(C), join(C), locale(M), uniq(C)

Diagnostics

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorders discovered under the **-c** option. When the last line of an input file is missing a **newline** character, *sort* appends one, prints a warning message, and continues.

Standards Conformance

sort is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

spell, hashmake, spellin, hashcheck

finds spelling errors

Syntax

spell [**-v**] [**-b**] [**-x**] [**-l**] [**-i**] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

Description

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most *troff*(CT), *tbl*(CT), and *eqn*(CT) constructions.

Under the **-v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **-b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*.

Under the **-x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(CT)) follows chains of included files (**.so** and **.nx** *troff*(CT) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **-l** option, *spell* will follow the chains of *all* included files. Under the **-i** option, *spell* will ignore all chains of included files.

Under the *+local_file* option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, it is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy-y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash codes on the standard output.

spellin n Reads *n* hash codes from the standard input and writes a compressed, or hashed *spelling_list*, such as */usr/lib/spell/hlista* or */usr/lib/spell/hlistb* , on the standard output. Information about the hash coding is printed on standard error.

hashcheck Reads a compressed, or hashed *spelling_list*, such as */usr/lib/spell/hlista* or */usr/lib/spell/hlistb* , and recreates the nine-digit hash codes for all the words in it, writing these codes on the standard output.

Examples

This example adds the words in *newwords* to the on-line dictionary (*/usr/lib/spell/hlista*):

```
cd /usr/lib/spell
cat newwords | ./hashmake | sort -u > newcodes
cat hlista | ./hashcheck > hashcodes
cat newcodes hashcodes | sort -u > newhash
cat newhash | ./spellin `cat newhash | wc -l` > hlist

mv hlista hlista.00
mv hlist hlista

cd /usr/dict
cat newwords words | sort -du > tempwords
mv words words.00
mv tempwords words
```

Remember to remove all temporary files after you are sure everything works.

The following example removes words from the on-line dictionary. You should first make a copy of **/usr/dict/words** that does not have the words you want to remove. Make sure the file is sorted in alphabetical order. Then, follow these steps:

```
cd /usr/lib/spell
cat /usr/dict/words | ./hashmake > hashcodes
cat hashcodes | ./spellin `cat hashcodes | wc -l` > newhlist

mv hlista hlista.00
mv newhlist hlista
```

Note that when you are manipulating large text, hash and hash code files, you should use **cat** (C) to open the files, since they may be extremely large.

Files

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

See Also

deroff(CT), eqn(CT), sed(C), sort(C), tbl(CT), tee(C), troff(CT)

Notes

The spelling list coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

By default, logging of errors to **/usr/lib/spell/spellhist** is turned off.

D_SPELL and S_SPELL can be overridden by placing alternate definitions in your environment.

Standards Conformance

hashcheck, *hashmake* and *spellin* are conformant with:
AT&T SVID Issue 2, Select Code 307-127.

spell is conformant with:
AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

spline

interpolates smooth curve

Syntax

spline [option] ...

Description

spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output has two continuous derivatives, and enough points to look smooth when plotted.

The following options are recognized, each as a separate argument.

-a Supplies abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

-k The constant k used in the boundary value computation

$$y_0'' = ky_1' , \ldots , y_n'' = ky_{n-1}'$$

is set by the next argument. By default $k = 0$.

-n Spaces output points so that approximately n intervals occur between the lower and upper x limits. (Default $n = 100$.)

-p Makes output periodic, i.e. matches derivatives at ends. First and last input values should normally agree.

-x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

Diagnostics

When data is not strictly monotone in *x, spline* reproduces the input without interpolating extra points.

Notes

A limit of 1000 input points is silently enforced.

split

splits a file into pieces

Syntax

split [-*n*] [file [name]]

Description

split reads *file* and writes it in as many *n*-line pieces as necessary (default 1000), onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if a dash (-) is given instead, the standard input file is used.

See Also

bfs(C), csplit(C)

Standards Conformance

split is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

strings

find the printable strings in an object file

Syntax

strings [-] [-o] [*-number*] filename ...

Description

strings looks for ASCII strings in a binary file. A string is any sequence of four or more printing characters ending with a newline or a null character. Unless the **-** flag is given, *strings* only looks in the initialized data space of object files. If the **-o** flag is given, then each string is preceded by its decimal offset in the file. If the *-number* flag is given then *number* is used as the minimum string length rather than 4.

strings is useful for identifying random object files and many other things.

See Also

hd(C), od(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

stty

sets the options for a terminal

Syntax

stty [**-a**] [**-g**] [options]

Description

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options. With the **-a** option, *stty* reports all of the option settings. The **-g** option causes stty to output the current stty settings of the terminal as a list of twelve hexadecimal numbers seperated by colons. This output may be used as a command line argument to stty to restore these settings later on. It is a more compact form than **stty -a**. For example, the following shell script uses **stty -g** to store the current stty settings, then turns off character echo while reading a line of input. The stored stty values are then restored to the terminal:

```
:
echo "Enter your secret code: \c"
old=`stty -g`
stty -echo intr ^a
read code
stty $old
```

The various modes are discussed in several groups that follow. Detailed information about the modes listed in the first four groups may be found in *termio*(M). *options* in the last group are implemented using *options* in the previous groups. Refer to *vidi*(C) for hardware specific information that describes control modes for the video monitor and other display devices.

Common Control Modes

parenb (-parenb)
Enables (disables) parity generation and detection.

parodd (-parodd)
Selects odd (even) parity.

cs5 cs6 cs7 cs8
Selects character size (see *termio*(M)).

0 Hangs up phone line immediately.

50 75 110 134 150 200 300 600
1200 1800 2400 4800 9600 19200
Sets terminal baud rate to the number given, if possible.

ispeed 50 75 110 134 150
1200 1800 3400 4800 9600 19200
sets terminal output baud rate separately.

ospeed 50 75 110 134 150
1200 1800 3400 4800 9600 19200
sets terminal input baud rate separately.

hupcl (-hupcl)
Hangs up (does not hang up) phone connection on last close.

hup (-hup)
Same as **hupcl (-hupcl)**.

cstopb (-cstopb)
Uses two(one) stop bits per character.

cread (-cread)
Enables (disables) the receiver.

clocal (-clocal)
Assumes a line without (with) modem control.

ctsflow (-ctsflow)
Enables (disables) CTS protocol for a modem or non-modem line.

rtsflow (-rtsflow)
Enables (disables) RTS signaling for a modem or non-modem line.

dtrflow (-dtrflow)
Enables (disables) DTR output flow control.

dsrflow (-dsrflow)
Enables (disables) DSR input flow control. Valid for non-modem lines only (i.e., when **clocal** is set.)

Input Modes

ignbrk (-ignbrk)
Ignores (does not ignore) break on input.

brkint (-brkint)
Signals (does not signal) INTERRUPT on break.

ignpar (-ignpar)
Ignores (does not ignore) parity errors.

parmrk (-parmrk)
Marks (does not mark) parity errors (see *tty*(M)).

inpck (-inpck)
Enables (disables) input parity checking.

istrip (-istrip)
Strips (does not strip) input characters to 7 bits.

inlcr (-inlcr)
Maps (does not map) NL to CR on input.

igncr (-igncr)
Ignores (does not ignore) CR on input.

icrnl (-icrnl)
Maps (does not map) CR to NL on input.

iuclc (-iuclc)
Maps (does not map) uppercase alphabetics to lowercase on input.

ixon (-ixon)
Enables (disables) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (-ixany)
Allows any character (only DC1) to restart output.

ixoff (-ixoff)
Requests that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (-opost)
Post-processes output (does not post-process output; ignores all other output modes).

olcuc (-olcuc)
Maps (does not map) lowercase alphabetics to uppercase on output.

onlcr (-onlcr)
Maps (does not map) NL to CR-NL on output.

ocrnl (-ocrnl)
Maps (does not map) CR to NL on output.

onocr (-onocr)
Does not (does) output CRs at column zero.

onlret (-onlret)
On the terminal NL performs (does not perform) the CR function.

ofill (-ofill)
Uses fill characters (use timing) for delays.

ofdel (-ofdel)
Fill characters are DELETEs (NULs).

cr0 cr1 cr2 cr3
Selects style of delay for RETURNs (see *tty*(M)).

nl0 nl1
Selects style of delay for LINEFEEDs (see *tty*(M)).

tab0 tab1 tab2 tab3
Selects style of delay for horizontal TABs (see *tty*(M)).

bs0 bs1
Selects style of delay for BACKSPACEs (see *tty*(M)).

ff0 ff1
Selects style of delay for FORMFEEDs (see *tty*(M)).

vt0 vt1
Selects style of delay for Vertical TABs (see *tty*(M)).

Local Modes

isig (-isig)
Enables (disables) the checking of characters against the special control characters INTERRUPT and QUIT.

icanon (-icanon)
Enables (disables) canonical input (ERASE and KILL processing).

xcase (-xcase)
Canonical (unprocessed) upper/lowercase presentation.

echo (-echo)
Echoes back (does not echo back) every character typed.

echoe (-echoe)
Echoes (does not echo) ERASE character as a SPACEBAR string. Note: this mode will erase the ERASE character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, TABs, and BACKSPACEs.

echok (-echok)
Echoes (does not echo) NL after KILL character.

lfkc (-lfkc)
The same as **echok (-echok)**; obsolete.

echonl (-echonl)
Echoes (does not echo) NL.

noflsh (-noflsh)
Disables (enables) flush after INTERRUPT or QUIT.

iexten (-iexten)
Enables extended implementation (implementation-defined) function.

tostop (-tostop)
Disables/enables background process group to write to controlling terminal only if job control is supported.

Control Assignments

control-character C
control-character C Sets *control-character* to *C*, where *control-character* is erase, kill, interrupt, quit, eof, eol, switch, or susp. If *C* is preceded by a caret (^) (escaped from the shell), then the value used is the corresponding CTRL character (e.g., ''^D'' is a CTRL-D); ''^?'' is interpreted as DELETE and ''^-'' is interpreted as undefined.

min *i*, **time** *i* (0<*i*<127)
When **-icanon** is set, and one character has been received, read requests are not satisfied until at least **min** characters have been received or the timeout value **time** has expired and one character has been received. See *tty*(C).

line *i*
Sets the line discipline to *i* (0 < *i* < 127). There are currently no line disciplines implemented.

Combination Modes

evenp or **parity**
Enables **parenb** and **cs7**.

oddp
Enables **parenb**, **cs7**, and **parodd**.

-parity, **-evenp**, or **-oddp**
Disables **parenb**, and sets **cs8**.

raw (**-raw** or **cooked**)
Enables (disables) raw input and output (no ERASE, KILL, INTERRUPT, QUIT, EOT, or output post-processing).

nl (**-nl**)
Unsets (sets) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**-lcase**)
Sets (unsets) **xcase**, **iuclc**, and **olcuc**.

LCASE (**-LCASE**)
Same as **lcase** (**-lcase**).

tabs (**-tabs** or **tab3**)
Preserves (expands to spaces) tabs when printing.

ek Resets ERASE and KILL characters back to normal CTRL-H and CTRL-U .

sane
Resets all modes to some reasonable values. Useful when a terminal's settings have been hopelessly scrambled.

term
Sets all modes suitable for the terminal type, TERM, where TERM is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

See Also

console(M), ioctl(S), vidi(C), tty(M), termio(M), termios(M)

Notes

Many combinations of options make no sense, but no checking is performed.

Standards Conformance

stty is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

su

makes the user a super-user or another user

Syntax

su [-] [name [arg ...]]

Description

su allows authorized users to become another user without logging off. The default user *name* is **root** (i.e., super-user).

su cannot be used to simply assume the login of another user in this implementation of UNIX. Instead, *su* can be used under four circumstances:

- The super-user can "su" to any account.
- An administrative user with the **su** authorization can "su" to the super-user account.
- A user can "su" to their own account (silly, but possible).
- A system daemon can "su" to an account.

To use *su*, the appropriate password must be supplied (unless you are already a super-user). If the password is correct, *su* will execute a new shell with the effective user ID set to that of the specified user. (The LUID is not changed.) The new shell will be the optional program named in the shell field of the specified user's password file (**/bin/sh** if none is specified (see *sh*(C)). To restore normal user ID privileges, press **EOF** (Ctrl-D) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(C), an *arg* of the form **-c** *string* executes *string* via the shell and an arg of **-r** gives the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(C). If the first argument to *su* is a **-**, the environment is changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is **-**, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **-sh** or **-su** to determine if it was invoked by *login*(M) or *su*(C), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(M) and *su*(C).

The file /etc/default/su can be used to control several aspects of how *su* is used. Several entries can be placed in /etc/default/su:

SULOG Name of log file to record all attempts to use *su*. Usually /usr/adm/sulog. If not set, no logfile is kept. (See example below.)

PATH The PATH environment variable to set for non-root users. If not set, it defaults to ":/bin:/usr/bin." The current PATH environment variable is ignored.

SUPATH When invoked by root, the path is set by default to "/bin:/usr/bin:/etc", unless this variable is defined., The current PATH is ignored.

CONSOLE Attempts to use *su* are logged to the named *device*, independently of SULOG.

For example, if you want to log all attempts by users to become root, create the file /etc/default/su. In this file, place a string similar to: SULOG=/usr/adm/sulog This causes all attempts by any user to switch user IDs to be recorded in the file **/usr/adm/sulog.** This filename is arbitrary. The *su* logfile records the original user, the UID of the su attempt, and the time of the attempt. If the attempt is successful, a plus sign (+) is placed on the line describing the attempt. A minus sign (-) indicates an unsuccessful attempt.

Examples

To become user **bin** while retaining your previously exported environment, enter:

su bin

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, enter:

su - bin

To execute *command* with the temporary environment and permissions of user **bin**, enter:

su - bin -c "*command args*"

Files

/etc/passwd	The system password file
/etc/default/su	Optional file containing control options
/etc/profile	The system profile
$HOME/.profile	The user profile

See Also

env(C), environ(M), login(M), passwd(F), profile(M), sh(C), auths(C)

Standards Conformance

su is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

sum

calculates checksum and counts blocks in a file

Syntax

sum [**-r**] file

Description

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of 512-byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over a transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

See Also

cmchk(C), machine(HW), wc(C)

Diagnostics

"Read error" is indistinguishable from end-of-file on most devices; check the block count.

Standards Conformance

sum is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

swconfig

produces a list of the software modifications to the system

Syntax

swconfig **[-a]** **[-p]**

Description

swconfig displays the modifications to the system software since its initialization, in much the same way that *hwconfig* tells the user what hardware is installed on the system. The program can tell the user what sets have been installed or removed from the system, as well as what release and what parts of the packages were installed at that time.

Options

Additional flags let the user ask to see all of the description of each installation on the system.

The default behavior is simple so that the information is displayed quickly. Additional flags can be used to perform more complex manipulations. Updates are recognized and noted as such. The release number is displayed in all cases.

Without options, *swconfig* generates a display similar to the following example:

```
Set                                 Release   Notes
---                                 -------   -----
Operating System                    2.3.1a    partially removed
International XENIX O.S. Supplem   2.0.0e    partially installed
Development System                  2.3.0b    removed
```

-a The **-a** flag lists all the information contained in /usr/lib/custom/history, but sorted by date. It groups products that were installed at the same time, but displays entries in reverse chronological order.

-p The flag **-p** is used to display package information in addition to the default information. A list of all the packages in a set is stored and their installed status tracked by the sequence of information in **/usr/lib/custom/history**.

Examples

Here is a sample output using the **-a** option:

```
Set: Operating System (prd = xos)
  Fri Mar 17 07:51:02 PST 1989
  removed successful              Release 2.3.1a      Type: 386GT
  Packages: HELP MOUSE

  Fri Mar 17 10:43:09 PST 1989
  removed successful              Release 2.3.1a      Type: 386GT
  Packages: VSH

Set: International XENIX O.S. Supplement (prd = sup.os)
  Fri Dec 16 10:32:53 PST 1988
  installed successful            Release 2.0.0e      Type: n286
  Packages: RTSUP BASE SYSADM FILE

  Fri Dec 16 11:03:37 PST 1988
  installed successful            Release 2.0.0e      Type: n286
  Packages: MAPFILE
```

Here is a sample output generated by the **-p** option:

```
Set                                Release  Notes      Packages
---                                -------  -----      --------
Operating System                   2.3.1a   removed    HELP MOUSE
Operating System                   2.3.1a   removed    VSH
International XENIX O.S. Supplem   2.0.0e   installed  RTSUP BASE
                                                        SYSADM FILE
International XENIX O.S. Supplem   2.0.0e   installed  MAPFILE
Develoment System                  2.3.0b   removed    ALL
```

See Also

custom(ADM)

tabs

set tabs on a terminal

Syntax

tabs [tabspec] [-Ttype] [+mn]

Description

The *tabs* command sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (*-code*), repetitive (*-n*), arbitrary (*n1,n2,...*), and file (*--file*). If no *tabspec* is given, the default value is **-8**, i.e., "standard" Altos UNIX System V tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

-a 1,10,16,36,72
Assembler, IBM S/370, first format

-a2 1,10,16,40,72
Assembler, IBM S/370, second format

-c 1,8,12,16,20,55
COBOL, normal format

-c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows [see *fspec*(F)]:

<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **-c2**. This is the recommended

format for COBOL. The appropriate format specification is [see *fspec*(F)]:

<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

-n A *repetitive* specification requests tabs at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value **8**: this represents the ''standard'' Altos UNIX System V tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0**, implying no tabs at all.

n1,n2,... The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1,10,20,30,** and **1,10,+10,+10** are considered identical.

--file If the name of a *file* is given, *tabs* reads the first line of the file, searching for a format specification [see *fspec*(F)]. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(C) command:

tabs -- file; **pr** file

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

-T*type* *tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term*(M). If no **-T** flag is supplied, *tabs* uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* [see *environ*(M)], *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is **10**. For a TermiNet, the first value in the tab list should be **1**, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

Examples

tabs -a example using *-code* (*canned* specification) to set tabs to the settings required by the IBM assembler:

columns 1, 10, 16, 36, 72.

tabs -8 example of using *-n* (*repetitive* specification), where *n* is **8**, causes tabs to be set every eighth position:

1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

tabs 1,8,36 example of using *n1*,*n2*,... (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs --$HOME/fspec.list/att4425
example of using *--file* (*file* specification) to indicate that tabs should be set according to the first line of **$HOME/fspec.list/att4425** [see *fspec*(F)].

Diagnostics

illegal tabs when arbitrary tabs are ordered incorrectly

illegal increment when a zero or missing increment is found in an arbitrary specification

unknown tab code when a *canned* code cannot be found

can't open if *--file* option used and file can't be opened

file indirection if *--file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted

See Also

newform(C), pr(C), tput(C), fspec(F), terminfo(F), environ(M), term(M)

Notes

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

The *tabspec* used with the *tabs* command is different from the one used with the *newform*(C) command. For example, **tabs -8** sets every eighth position; whereas **newform -i-8** indicates that tabs are set every eighth position.

Standards Conformance

tabs is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

tail

displays the last part of a file

Syntax

tail [±[number][**lbc**] [**-f**]] [file]

Description

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *-number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **-f** (''follow'') option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f file
```

will print the last ten lines of *file*, followed by any lines that are appended to *file* between the time *tail* is initiated and killed.

See Also

dd(C)

Notes

Tails relative to the end of the file are kept in a buffer, and thus are limited to approximately 300 lines. Unpredictable results can occur if character special files are ''tailed''.

Standards Conformance

tail is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

and The X/Open Portability Guide II of January 1987.

tape, mcart

magnetic tape maintenance program

Syntax

tape [**-c**] [**-f**] [**-a arg**] **command** [**device**]

mcart command [device]

Description

tape sends commands to and receives status from the tape subsystem. *tape* can communicate with QIC-02 cartridge tape drives, SCSI tape drives, and QIC-40, QIC-80 and Irwin mini-cartridge tape drives. (The *mcart* program is automatically invoked by *tape* when options specific to the Irwin driver are used.)

tape reads **/etc/default/tape** to find the default device name for sending commands and receiving status. For example, the following line in **/etc/default/tape** will cause *tape* to communicate with the QIC-02 cartridge tape device:

device = /dev/xct0

If a device name is specified on the command line, it overrides the default device. *tape* queries the device to determine its device type. If the device does not respond to the query, for example if the cartridge tape driver is from an earlier release, *tape* will print a warning message and assume the device is a QIC-02 cartridge tape.

You can explicitly specify the type of the device by using the device type flags, as follows:

-c	QIC-02 cartridge tape
-s	SCSI tape
-f	QIC-40 mini-cartridge tape
-8	QIC-80 mini-cartridge tape
-i	Irwin mini-cartridge tape

The **-a** flag allows you to pass an argument to commands that can use them. The only command that currently can take an argument is the **format** command, and a **format** argument is only valid with QIC-40 and QIC-80 tape drives.

The following commands can be used with the various tape drivers supported under UNIX. The letters following each description indicate which drivers support each command:

A	All drivers
C	QIC-02 cartridge tape driver
S	SCSI tape driver
F	QIC-40 and QIC-80 mini-cartridge tape drivers
I	Irwin mini-cartridge tape driver

amount
Report amount of data in current or last transfer. (C,S,F)

erase
Erase and retension the tape cartridge. (C,S,F)

reset
Reset tape controller and tape drive. Clears error conditions and returns tape subsystem to power-up state. (C,S,F)

reten
Retension tape cartridge. Should be used periodically to remedy slack tape problems. Tape slack can cause an unusually large number of tape errors. (A)

rewind
Rewind to beginning of tape. (A)

status
The status output looks like this:

`status:`	*status message*
`soft errors:`	*n*
`underruns:`	*m*

status message is a report of the current status of the drive; ''no cartridge,'' ''write protected,'' or ''beginning of tape'' are typical status messages.

soft errors is the number of recoverable errors that occurred during the last tape operation. A recoverable error is one which is correctable by the drive or controller. An example of a non-recoverable ''hard'' error is an attempt to write to a write-protected cartridge. Note that if the number of soft errors greatly exceeds the manufacturer's specifications, the drive may require service or replacement.

underruns is the number of times the tape drive had to stop and restart due to tape buffer underflows. Underruns are not an error indication, but that the data transfer did not occur at the drive's maximum data transfer rate. The number of overruns can be affected by system load. (C,S,F)

format

Format the tape cartridge. Tapes must be formatted before they can be used. This command takes approximately one minute per megabyte of tape capacity. Note that on Irwin mini-cartridge tape drives, blank tapes must be servo-written with the **servo** command before they can be formatted. If an argument is provided with the **-a** flag, the number of tracks specified by the argument will be formatted. Only even numbers less than or equal to the number of tracks on the tape are allowed. (See **tape**(HW) for more information.) If no argument is given, the entire tape will be formatted. (F,I)

getbb

Prints a list of bad tape blocks detected during the last tape operation. This listing can be saved in a file for use by the **putbb** command. (F)

map

Prints out a map of the bad blocks on the tape. The format is a series of lines of the format:

```
track n: -------------X------...
```

Each '-' represents a good block on the track; an 'X' represents a block marked as bad. (F,I)

putbb

Reads a list of bad tape blocks from the standard input and adds them to the bad block table on the tape. The format expected by **putbb** is the same as generated by the **getbb** command. (F)

rfm

Wind tape forward to the next file mark. (C,S)

wfm

Write a file mark at the current tape position. (C,S)

Irwin-specific Commands

The following commands are all specific to Irwin drives.

drive

displays information about the Irwin driver and the tape drive. An example display is:

```
Special file: /dev/rmc0
Driver version: 1.0.6a
Drive type: 285XL
Drive firmware: A0
Controller type: SYSFDC
Unit select (0-3): 3
```

Special file is the name of the special file used to access the driver.

Driver version is the version of the driver linked with the kernel.

Drive type is an "equivalent" tape drive model number as determined by the MC driver. Since the exact model number of the tape drive depends on the drive's form factor and whether the drive is mounted in its own cabinet, the equivalent model number may not be the exact model of the installed tape drive. The following is a list of equivalent drives:

110: 110, 310, 410
120[XL]: 120, 220, 320, 420, 720, 2020
125: 125, 225, 325, 425, 725
145[XL]: 145, 245, 345, 445, 745, 2040
165: 165, 265, 465, 765
285XL: 285, 485, 785, 2080
287XL: 287, 487, 787, 2120

The brackets in the 120[XL] and 145[XL] mean the letters "XL" may or may not be present. When the letters "XL" appear, the drive is capable of servo writing extra long (i.e., 307.5 foot DC2120) tapes.

Note: When this field displays "125/145," either a 125 drive or an early model 145 drive with a DC1000 is present, the driver can't distinguish between the two. A 125 drive will only accept a DC1000 cartridge (a DC2000 or DC2120 will not fit). A 145 drive will accommodate DC1000, DC2000, or DC2120 cartridges.

Drive firmware is the firmware part number and revision level. This line is present only for drives which report this information.

Controller type: is a mnemonic for the floppy controller to which the tape drive is attached:

Mnemonic	Description
SYSFDC	System floppy controller
ALTFDC	Alternate floppy controller
4100MC	Irwin 4100MC Micro Channel controller
4100MCB	Second 4100MC Micro Channel controller
4100	Irwin 4100 PC Bus controller
4100B	Second 4100 PC Bus controller

Unit select (0-3) gives the controller's unit select, in the range 0 through 3. The unit select selects the drive.

servo

Prepares a blank tape for formatting by writing servo information on each track. This command must be used on blank mini-cartridge tapes before they can be used in an Irwin mini-cartridge drive. If the tape has been previously servo-written, it must be bulk-erased with a commercial tape eraser before it can be servo-written again. Normally, a tape should only be servo-written once in its lifetime, although it can be formatted with the **format** command many times.

info

displays Irwin cartridge information. For example:

```
Cartridge state: Formatted
Cartridge format: 145
Write protect slider position: RECORD
```

Cartridge state is the current state of the cartridge's format.

Cartridge format indicates the format on the cartridge's tape. The format is given in a code which is the same as the drive model on which the cartridge was originally formatted (see **drive** and *tape*(HW) for details). When the cartridge is blank, the code has the format which would be applied by the **format** command.

Write protect slider position is RECORD or PROTECT.

capacity

cartridge capacity in 512-byte blocks.

kapacity

cartridge capacity in 1024-byte blocks.

These two commands give the total usable data storage capacity of a formatted tape cartridge. Variations in cartridge capacity are due to differing numbers of bad blocks.

Files

/dev/rStp0	/dev/rct0	/dev/erct0	/dev/rmc1
/dev/nrStp0	/dev/nrct0	/dev/xct0	/dev/mcdaemon
/dev/xStp0	/dev/rct2	/dev/rctmini	
/dev/rft0	/dev/nrct2	/dev/xctmini	
/dev/xft0	/dev/xct0	/dev/rmc0	

/etc/default/tape

Include files:

/usr/include/sys/tape.h
/usr/include/sys/ct.h
/usr/include/sys/ft.h
/usr/include/sys/ir.h

See Also

backup(ADM), cpio(C), dd(C), restore(ADM), tape(HW), tar(C), xbackup(ADM), xrestore(ADM)

Notes

See *tape*(HW) for a list of supported tape drives.

The **amount** and **reset** commands can be used while the tape is busy with other operations. All other commands wait until the currently executing command has been completed before proceeding.

When you are using the non-rewinding tape device or the *tape* commands **rfm** and **wfm**, the tape drive light remains on after the command has been completed, indicating that more operations may be performed on the tape. The *tape* **rewind** command may be used to clear this condition.

For more information on devicefiles, (listed above), see the *tape*(HW) manual page.

If you use the **status** command while the tape drive is busy, no message is displayed until the drive is free.

The **amount** command doesn't work with QIC-40 mini-cartridge tape devices.

tapecntl

AT&T tape control for QIC-24/QIC-02 tape device

Syntax

tapecntl [**-etrw**] [**-p arg**]

Description

tapecntl will send the optioned commands to the tape device driver sub-device **/dev/rmt/c0s0** for all commands except "position," which will use sub-device **/dev/rmt/c0s0n** using the *ioctl* command function. Sub-device **/dev/rmt/c0s0** provides a rewind on close capability, while **/dev/rmt/c0s0n** allows for closing of the device without rewind. Error messages will be written to standard error.

-e	erase tape
-t	retension tape
-r	reset tape device
-w	rewind tape
-p[*n*]	position tape to "end of file" mark - *n*

Erasing the tape causes the erase bar to be activated while moving the tape from end to end, causing all data tracks to be erased in a single pass over the tape.

Retensioning the tape causes the tape to be moved from end to end, thereby repacking the tape with the proper tension across its length.

Reset of the tape device initializes the tape controller registers and positions the tape at the beginning of the tape mark (BOT).

Rewinding the tape will move the tape to the BOT.

Positioning the tape command requires an integer argument. Positioning the tape will move the tape forward relative to its current position to the end of the specified file mark. The positioning option used with an argument of zero will be ignored. Illegal or out-of-range value arguments to the positioning command will leave the tape positioned at the end of the last valid file mark.

Options may be used individually or strung together with selected options being executed sequentially from left to right in the command line.

Files

/usr/lib/tape/tapecntl
/dev/rmt/c0s0n
/dev/rmt/c0s0

Notes

Exit codes and their meanings are as follows:

exit (1) device function could not initiate properly due to misconnected cables or poorly inserted tape cartridge.

exit (2) device function failed to complete properly due to unrecoverable error condition, either in the command setup or due to mechanical failure.

exit (3) device function failed due to the cartridge being write protected or to the lack of written data on the tape.

exit (4) device **/dev/rmt/c0s0n** or **/dev/rmt/c0s0** failed to open properly due to already being opened or claimed by another process.

tapedump

dumps magnetic tape to output file

Syntax

tapedump [-a|-e] [-o|-h] [-btsn*num*] *tape_device output_file*

Description

tapedump dumps the contents of magnetic tapes according to the options specified. Options include conversion from input format to user specified output format, specification of input and output blocksize, and the ability to specify that the dump begin at a specific start block on the tape and proceed for a specified number of blocks.

Options

Option	Value
tape_device	The input tape device.
-a	Convert from EBCDIC input to ASCII output.
-e	Convert from ASCII input to EBCDIC output.
-o	Display tape output in octal format.
-h	Display tape output in hexadecimal format.
-b *num*	skips *n* input records before starting dump.
-t *num*	Specify which tape file to begin dump from, where *num* is the tape file sequence number.
-s *num*	Specify tape block address from which to start dump.
-n *num*	Specify dump of only *num* blocks.
output_file	The output filename; standard output is the default.

Examples

This command reads a tape starting at block 400 and outputs the results in hexadecimal format into a user specified file called **/tmp/hex.dump**:

tapedump -b*400* **-h** */dev/rct0 /tmp/hexdump*

This command reads an EBCDIC tape and converts the standard output to ASCII:

tapedump -a */dev/rct0*

See Also

sysadmsh(ADM), dd(C), hd(C), od(C), tape(C)

Notes

The output file may be specified to be another tape device.

tar

archives files

Syntax

tar [key] [files]

Description

tar saves and restores files to and from an archive medium, which is typically a storage device such as floppy disk or tape, or a regular file. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Valid function letters are **c**, **t**, **x**, and **e**. Other arguments to the command are *files* (or directory names) specifying which files are to be backed up or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory. The **r** and **u** option cannot be used with tape devices.

The function portion of the key is specified by one of the following letters:

r The named *files* are written to the end of an existing archive.

x The named *files* are extracted from the archive. If a named file matches a directory whose contents had been written onto the archive, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire contents of the archive are extracted. Note that if several files with the same name are on the archive, the last one overwrites all earlier ones.

t The names of the specified files are listed each time that they occur on the archive. If no *files* argument is given, all the names on the archive are listed.

u The named *files* are added to the archive if they are not already there, or if they have been modified since last written on that archive.

c Creates a new archive; writing begins at the beginning of the archive, instead of after the last file.

The following characters may be used in addition to the letter that selects the desired function:

0,...,9999
This modifier selects the drive on which the archive is mounted. The default is found in the file **/etc/default/tar**.

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to display the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the archive entries than just the name.

w Causes *tar* to display the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with y is given, the action is performed. Any other input means "no".

f Causes *tar* to use the next argument as the name of the archive instead of the default device listed in **/etc/default/tar**. If the name of the file is a dash (-), *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *tar* can also be used to move hierarchies with the command:

cd fromdir; tar cf - . | (cd todir; tar xf -)

b Causes *tar* to use the next argument as the blocking factor for archive records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).

F Causes *tar* to use the next argument as the name of a file from which succeeding arguments are taken.

l Tells *tar* to display an error message if it cannot resolve all of the links to the files being backed up. If **l** is not specified, no error messages are displayed.

m Tells *tar* to not restore the modification times. The modification time of the file is the time of extraction.

k Causes *tar* to use the next argument as the size of an archive volume in kilobytes. The minimum value allowed is 250. Very large files are split into "extents" across volumes. When restoring from a multivolume archive, *tar* only prompts for a new volume if a split file has been partially restored. To override the value of **k** in the **default** file, specify **k** as 0 on the command line.

e Prevents files from being split across volumes (tapes or floppy disks). If there is not enough room on the present volume for a given file, *tar* prompts for a new volume. This is only valid when the **k** option is also specified on the command line.

n Indicates the archive device is not a magnetic tape. The **k** option implies this. Listing and extracting the contents of an archive are sped because *tar* can seek over files it wishes to skip. Sizes are printed in kilobytes instead of tape blocks.

p Indicates that files are extracted using their original permissions. It is possible that a non-super-user may be unable to extract files because of the permissions associated with the files or directories being extracted.

A Suppresses absolute filenames. Any leading ''/''characters are removed from filenames. During extraction arguments given should match the relative (rather than the absolute) pathnames. With the **c**, **r**, **u** options the **A** options can be used to inhibit putting leading slashes in the archive headers.

tar reads **/etc/default/tar** to obtain default values for the device, blocking factor, volume size, and the device type (tape or non-tape). If no numeric key is specified on the command, *tar* looks for a line in the **default** file beginning with the string *archive=*. Following this pattern are 4 blank separated strings indicating the values for the device, blocking factor, volume size and device type, in that order. A volume size of '0' indicates infinite volume length. This entry should be modified to reflect the size of the tape volumes used.

For example, the following is the default device entry from **/etc/default/tar** :

```
archive=/dev/fd096ds15 10 1200 n
```

The *n* in the last field, means that this device is not a tape. Use *y* for tape devices. Any default value may be overridden on the command line. The numeric keys (0-7) select the line from the default value beginning with *archive#=*, where # is the numeric key. When the **f** key letter is specified on the command line, the entry *"archivef="* is used. In this case, the default file entry must still contain 4 strings, but the first entry (specifying the device) is not significant. The default file **/etc/default/tar** need not exist if a device is specified on the command line.

Notes

A critical consideration when creating a tar volume involves the use of absolute or relative pathnames. Consider the following *tar* command examples, as executed from the directory **/u/target**:

```
tar cv /u/target/arrow

tar cv arrow
```

The first command creates a tar volume with the *absolute* pathname: /u/target/arrow. The second yields a tar volume with a *relative* pathname: ./arrow. (The ./ is implicit and shown here as an example, ./ should not be specified when retrieving the file from the archive.) When restored, the first example results in the file **arrow** being written to the directory /u/target (if it exists and you have write permission) no matter what your working directory. The second example simple writes the file **arrow** to your present working directory.

Absolute pathnames specify the location of a file in relation to the root directory (/); relative pathnames are relative to the current directory. This must be taken into account when making a tar tape or disk. Backup volumes use absolute pathnames so that they can be restored to the proper directory. Use relative pathnames when creating a tar volume where absolute pathnames are unnecessary.

Examples

If the name of a floppy disk device is **/dev/fd1**, then a tar format file can be created on this device by entering:

```
assign /dev/fd
tar cvfk /dev/fd1 360 files
```

where *files* are the names of files you want archived and 360 is the capacity of the floppy disk in kilobytes. Note that arguments to key letters are given in the same order as the key letters themselves, thus the **fk** key letters have corresponding arguments **/dev/fd1** and **360**. Note that if a *file* is a directory, the contents of the directory are recursively archived. To display a listing of the archive, enter:

```
tar tvf /dev/fd1
```

At some later time you will likely want to extract the files from the archive floppy. You can do this by entering:

```
tar xvf /dev/fd1
```

The above command extracts all files from the archive, using the exact same pathnames as used when the archive was created. Because of this behavior, it is normally best to save archive files with relative pathnames rather than absolute ones, since directory permissions may not let you read the files into the absolute directories specified. (See the A flag under *Options*.)

In the above examples, the **v** verbose option is used simply to confirm the reading or writing of archive files on the screen. Also, a normal file could be substituted for the floppy device **/dev/fd1** shown in the examples.

Files

/etc/default/tar	Default devices, blocking and volume sizes, device type
/tmp/tar*	

Diagnostics

Displays an error message about bad key characters and archive read/write errors.

Displays an error message if not enough memory is available to hold the link tables.

Notes

There is no way to ask for the *n*th occurrence of a file.

tar does not verify the selected media type.

The **u** option can be slow.

The limit on filename length is 100 characters.

When archiving a directory that contains subdirectories, *tar* will only access those subdirectories that are within 17 levels of nesting. Subdirectories at higher levels will be ignored after *tar* displays an error message.

When using *tar* with a raw device, specify the block size with the **b** option as a multiple of 512 bytes. For example, to use a 9K block size, enter:

```
tar cvfb /dev/rfd0 18 file
```

Do not enter:

```
tar xfF - -
```

This would imply taking two things from the standard input at the same time.

Use error-free floppy disks for best results with *tar*.

Standards Conformance

tar is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

tee

creates a tee in a pipe

Syntax

tee [**-i**] [**-a**] [**-u**] [file] ...

Description

tee transcribes the standard input to the standard output and makes copies in the *files*. The **-i** option ignores interrupts; the **-a** option causes the output to be appended to the *files* rather than overwriting them. The **-u** option causes the output to be unbuffered.

Examples

The following example illustrates the creation of temporary files at each stage in a pipeline:

```
grep ABC | tee ABC.grep | sort | tee ABC.sort | more
```

This example shows how to tee output to the terminal screen:

```
grep ABC | tee /dev/ttyxx | sort | uniq >final.file
```

Standards Conformance

tee is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

test

tests conditions

Syntax

test expr

[expr]

Description

test evaluates the expression *expr*, and if its value is true, returns a zero (true) exit status; otherwise, *test* returns a nonzero exit status if there are no arguments. The following primitives are used to construct *expr*:

-r *file*	True if *file* exists and is readable.
-w *file*	True if *file* exists and is writable.
-x *file*	True if *file* exists and is executable.
-f *file*	True if *file* exists and is a regular file.
-d *file*	True if *file* exists and is a directory.
-c *file*	True if *file* exists and is a character special file.
-b *file*	True if *file* exists and is a block special file.
-u *file*	True if *file* exists and its set-user-ID bit is set.
-g *file*	True if *file* exists and its set-group-ID bit is set.
-k *file*	True if *file* exists and its sticky bit is set.
-s *file*	True if *file* exists and has a size greater than zero.
-t [*fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
-z *s1*	True if the length of string *s1* is zero.
-n *s1*	True if the length of string *s1* is nonzero.

s1 = *s2*	True if strings *s1* and *s2* are identical.
s1 != *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.
n1 -eq *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

!	Unary negation operator
-a	Binary *and* operator
-o	Binary *or* operator (**-a** has higher precedence than **-o**)
(expr)	Parentheses for grouping

Notice that all the operators and flags are separate arguments to *test*. Notice also, that parentheses are meaningful to the shell and, therefore, must be escaped.

See Also

find(C), sh(C)

Warning

In the second form of the command (i.e., the one that uses [ ], rather than the word *test*), the square brackets must be delimited by blanks.

Standards Conformance

test is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

tic

terminfo compiler

Syntax

tic [**-v** [*n*] [**-p** permlist]] file ...

Description

tic translates terminfo files from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo.**

The **-v** (verbose) option causes *tic* to output trace information showing its progress. If the optional digit *n* is appended, the level of verbosity can be increased.

The **-p** option directs *tic* to create a permissions file **permlist** for use with *fixperm*(ADM).

tic compiles all terminfo descriptions in the given files. When a **use=** field is discovered, *tic* first searches the current file and then the master file **./terminfo.src.**

If the environment variable TERMINFO is set, the results are placed there instead of **/usr/lib/terminfo.**

Some limitations: the total size of a description cannot exceed 4096 bytes; the name field cannot exceed 128 bytes.

Files

/usr/lib/terminfo/*/* -Compiled terminal capability database.

See Also

terminfo(M), terminfo(S), terminfo(F)

Standards Conformance

tic is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

time

times a command

Syntax

time command

Description

The given *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on the standard error.

See Also

times(S)

Standards Conformance

time is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

touch

updates access and modification times of a file

Syntax

touch [**-amc**] [mmddhhmm[yy]] files

Description

touch causes the access and modification times of each argument to be updated. If no time is specified (see *date*(C)) the current time is used. If a new file is created using *touch*, the modification and access times can be set to any time. However, the creation time is automatically set to the current time at the time of creation, and cannot be changed. The first *mm* refers to the month, *dd* refers to the day, *hh* refers to the hour, the second *mm* refers to the minute, and *yy* refers to the year. The **-a** and **-m** options cause touch to update only the access or modification times respectively (default is **-am**). The **-c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

See Also

date(C), utime(S)

Standards Conformance

touch is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

tput

queries the terminfo database

Syntax

tput [**-T***type*] attribute

Description

The command *tput* uses the terminfo database to make the values of terminal-dependent *attributes* available to the shell. *tput* outputs a string if the terminal *attribute* is of type string, or an integer if the *attribute* is of type integer. If the *attribute* is of type Boolean, *tput* simply sets the exit code (0 for true if the terminal has the capability, 1 for false if it does not) and produces no output.

The **-T** flag indicates the type of the terminal. Normally this option is unnecessary, as the default is taken from the environment variable **TERM**.

attribute is the terminal capability name from the terminfo database.

Examples

tput clear Echo clear-screen sequence for the current terminal.

tput cols Print the number of columns for the current terminal.

tput -T450 cols Print the number of columns for the 450 terminal.

bold='tput smso'
offbold='tput rmso' Set the shell variables "bold" to begin standout mode sequence and "offbold" to end standout mode sequence for the current terminal. This might be followed by a prompt, such as:

```
echo "${bold}Name: ${offbold}\c"
```

tput hc — Set exit code to indicate if the current terminal is a hardcopy terminal.

Files

/usr/lib/terminfo/*/* -Compiled terminal capability database.

See Also

terminfo(M), terminfo(S), tic(C), stty(C)

Notes

If the *attribute* is of type boolean, a value of 0 is returned for TRUE and a value of 1 for FALSE.

If the *attribute* is of type string or integer, a value of 0 is returned upon successful completion. Any other value returned indicates an error. For example, the specification of a bad *attribute* (any capability name that is not found in the terminfo database) produces an error.

Standards Conformance

tput is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

tr

translates characters

Syntax

tr [**-cds**] [string1 [string2]]

Description

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **-cds** may be used:

-c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal

-d Deletes all input characters in *string1*

-s Squeezes all strings of repeated output characters that are in *string2* to single characters

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a-z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a**n*] Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits, stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in *file1*, one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline:

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

See Also

ed(C), sh(C), ascii(M)

Notes

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

Standards Conformance

tr is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

translate

translates files from one format to another

Syntax

translate option [infile] [outfile]

Description

translate translates files according to the options specified. Translation is done according to the options defined below.

format is assumed to be a file in the directory **/usr/lib/mapchan/translate** if a full pathname is not provided.

translate uses standard input and standard output unless otherwise specified via the optional filename arguments.

Options

-ea	From EBCDIC to ASCII.
-ae	From ASCII to EBCDIC.
-fe *format*	From a user defined format to EBCDIC format.
-fa *format*	From a user defined format to ASCII format.
-ef *format*	From EBCDIC format to a user defined format.
-af *format*	From ASCII format to a user defined format.
-bm	From binary/object code to mailable ASCII *uuencode* format.
-mb	From mailable ASCII *uuencode* format to original binary.

Files

/usr/lib/mapchan/translate/*

See Also

uuencode(C), dd(C), mapchan(M), sysadmsh(ADM)

Notes

The **-bm** and **-mb** options are, for example, used to translate executable object code format to ASCII for transfer across communications networks.

The syntax for the user defined format file is the same as the syntax for the mapping files for *mapchan*(M) and *trchan*.

Use **dd** to convert character and file formats (especially tapes) to the format specified. Example:

```
dd if=/dev/rmt0 of=outfile ibs=800 cbs=80 conv=ascii,lcase
```

This command reads an EBCDIC tape, blocked ten 80-byte EBCDIC card images per record, into the ASCII file *outfile*. For more information on conversion options, refer to **dd**(C) in the *User's Reference*.

true

returns with a zero exit value

Syntax

true

Description

true does nothing except return with a zero exit value. *false*(C), *true*'s counterpart, does nothing except return with a nonzero exit value. *true* is typically used in shell procedures such as:

```
while true
do
        command
done
```

See Also

sh(C), false (C)

Diagnostics

true has exit status zero.

Standards Conformance

true is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

tset

provide information to set terminal modes

Syntax

tset [*options*] [*type*]

Description

tset allows the user to set a terminal's ERASE and KILL characters, and define the terminal's type and capabilities by creating values for the TERM environment variable. *tset* initializes or resets the terminal with *tput*(C). If a *type* is given with the **-s** option, *tset* creates information for a terminal of the specified type. The type may be any type given in the *terminfo* database. If the *type* is not specified with the **-s** option, *tset* creates information for a terminal of the type defined by the value of the environment variable, TERM, unless the **-h** or **-m** option is given. If the **TERM** variable is defined, *tset* uses the *terminfo* database entry. If these options are used, *tset* searches the **/etc/ttytype** file for the terminal type corresponding to the current serial port; it then creates information for a terminal based on this type. If the serial port is not found in **/etc/ttytype**, the terminal type is set to **unknown**.

tset displays the created information at the standard output. The information is in a form that can be used to set the current environment variables. The exact form depends on the login shell from which *tset* was invoked. The following examples illustrate how to use this information to change the variables.

There are the following options:

-e[*c*]
: Sets the ERASE character to *c* on all terminals. The default setting is the BACKSPACE, or CTRL-H.

-E[*c*]
: Identical to the **-e** command except that it only operates on terminals that can BACKSPACE.

-k[*c*]
: Sets the KILL character to *c*, defaulting to CTRL-U.

\- Prints the terminal type on the standard output.

-s Outputs the "setenv" commands [for *csh*(C)], or "export" and assignment commands [for *sh*(1)]. The type of commands are determined by the user's login shell.

-h Forces *tset* to search **/etc/ttytype** for information and to overlook the environment variable, TERM.

-S Only outputs the strings to be placed in the environment variables, without the shell commands printed for **-S**.

-r Prints the terminal type on the diagnostic output.

-Q Suppresses the printing of the "Erase set to" and "Kill set to" messages.

-I Suppresses printing of the terminal initialization strings, e.g., spawns *tput reset* instead of *tput init*.

-m[*ident*][*test baudrate*]:*type*
Allows a user to specify how a given serial port is is to be mapped to an actual terminal type. The option applies to any serial port in **/etc/ttytype** whose type is indeterminate (e.g., *dialup*, *plugboard*, etc.). The *type* specifies the terminal type to be used, and *ident* identifies the name of the indeterminate type to be matched. If no *ident* is given, all indeterminate types are matched. The *test baudrate* defines a test to be performed on the serial port before the type is assigned. The *baudrate* must be as defined in *stty*(C). The *test* may be any combination of: >, =, <, @, and !. If the *type* begins with a question mark, the user is asked if he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. The question mark must be escaped to prevent filename expansion by the shell. If more than one **-m** option is given, the first correct mapping prevails.

tset is most useful when included in the **.login** [for *csh*(C)] or **.profile** [for *sh*(C)] file executed automatically at login, with **-m** mapping used to specify the terminal type you most frequently dial in on.

Examples

tset gt42

tset -mdialup\>300:adm3a -mdialup:dw2 -Qr -e#

tset -m dial:ti733 -m plug:\?hp2621 -m unknown:\? -e -k^U

To use the information created by the **-s** option for the Bourne shell, (*sh*), repeat these commands:

```
tset -s ... > /tmp/tset$$
/tmp/tset$$
rm /tmp/tset$$
```

To use the information created for *csh*, use:

```
set noglob
set term=('tset -S ....')
setenv TERM $term[1]
unset term
unset noglob
```

Files

/etc/ttytype	Port name to terminal type map database
/usr/lib/terminfo/*	Terminal capability database

See Also

stty(C), termio(M), tput(C), tty(M), terminfo(F)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

tty

gets the terminal's name

Syntax

tty [-s]

Description

The *tty* command prints the pathname of the user's terminal on the standard output. The **-s** option inhibits printing, allowing you to test just the exit code.

Exit Codes

0 if the standard input is a terminal, 1 otherwise.

Diagnostics

not a tty If the standard input is not a terminal and **-s** is not specified

Standards Conformance

tty is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

umask

sets file-creation mode mask

Syntax

umask [000]

Description

The user file-creation mode mask is set to *000*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively. Only the low-order 9 bits of *cmask* and the file mode creation mask are used. The value of each specified digit is ''subtracted'' from the corresponding ''digit'' specified by the system for the creation of any file (see *umask*(S) or *creat*(S)). This is actually a binary masking operation, and thus the name "umask". In general, binary ones remove a given permission, and zeros have no effect at all. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode 777 become mode 755 ; files created with mode 666 become mode 644).

If *000* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell. By default, login shells have a umask of 022.

umask is built in to *csh* and *sh*.

See Also

chmod(C), csh(C), sh(C), chmod(S), creat(S), umask(S)

Standards Conformance

umask is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

uname

prints the name of the current system

Syntax

uname [**-snrvma**]
uname [**-S** system name]

Description

The *uname* command prints the current system name of the Altos UNIX System V system on the standard output file. It is mainly useful to determine which system you are using. The options cause selected information returned by *uname*(S) to be printed:

-s print *system name* (default).

-n print *nodename* (the nodename is the name by which the system is known to a communications network).

-r print the operating system release.

-v print the operating system version.

-m print the machine hardware name.

-a print all the above information.

On your computer, the system name and the nodename may be changed by specifying a *system name* argument to the **-S** option. The *system name* argument is restricted to 8 characters. Only the super-user is allowed this capability.

See Also

uname(S)

Standards Conformance

uname is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

uniq

reports repeated lines in a file

Syntax

uniq [**-udc** [+n] [-n]] [input [output]]

Description

uniq reads the *input* file and compares adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(C). If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of nonspace, nontab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

See Also

comm(C), sort(C)

Standards Conformance

uniq is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

units

converts units

Syntax

units

Description

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division is shown by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, as well as the following:

pi Ratio of circumference to diameter

c Speed of light

e Charge on an electron

g Acceleration of gravity

force Same as g

mole
Avogadro's number

water
Pressure head per unit height of water

au Astronomical unit

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g. **lightyear**). British units that differ from their US counterparts are prefixed with "br". For a complete list of units, enter:

cat /usr/lib/unittab

Files

/usr/lib/unittab

uptime

displays information about system activity

Syntax

uptime

Description

uptime prints the current time of day, the length of time the system has been up, and the number of users logged onto the system. On systems that maintain the necessary data, load averages are also shown. Load averages are the number of processes in the run queue averaged over 1, 5, and 15 minutes. All of this information is also contained in the first line of the *w*(C) command.

See Also

w(C)

usemouse

maps mouse input to keystrokes for use with non-mouse based programs

Syntax

usemouse [**-f** conffile] [**-t** type] [**-h** horiz_sens] [**-v** vert_sens] [**-c** cmd] [**-b**] parameters

Description

This utility allows you to use a mouse with any program that would otherwise accept only keyboard input.

For example, you can use a mouse with *vi*(C) to move the cursor around the screen and generate your most commonly used *vi* commands. The *usemouse*(C) command translates mouse input into specific keystrokes required by a program. You can use any of several predefined mouse keystroke sets (called maps) that correspond to different popular programs. You can also define your own maps with keystrokes that match different mouse movements and mouse buttons.

Options

The options are:

-f conffile
The **-f** flag may be used to select an alternate configuration file. The alternate configuration file, *conffile*, should use the format of **/etc/default/usemouse** and be entered as an absolute pathname on the command line. For example:

usemouse -f /u/daniel/mouseconf

is the correct form to specify an alternate configuration file. The **-f** and **-t** flags are mutually exclusive.

-t type
The **-t** flag may be used to select a predefined configuration file. *type* can be the name of any file in **/usr/lib/mouse**, such as *vi*, *rogue*, or any others the system administrator chooses to place there. These files are identical in format to **/etc/default/usemouse**.

-h horiz_sens
Defines the horizontal sensitivity. Horizontal mouse movements smaller than this threshold are ignored. Mouse movements that are

multiples of this value generate multiple strings. The sensitivity defaults to 5 units. The minimum value is 1 unit, and the maximum is 100 units. The lower the value, the more sensitive your mouse is to motion. Note that setting a high value may cause your mouse to behave as though it is not functioning, due to the large motion required to generate a signal.

-v vert_sens
Defines the vertical sensitivity. Vertical mouse movements smaller than this threshold are ignored. Mouse movements that are multiples of this value generate multiple strings. The sensitivity defaults to 5 units. The minimum value is 1 unit, and the maximum is 100 units. The lower the value, the more sensitive your mouse is to motion. Note that setting a high value may cause your mouse to behave as though it is not functioning, due to the large motion required to generate a signal.

-c cmd
This option selects a command for *usemouse* to run. This defaults to the shell specified in the SHELL environment variable. If SHELL is unspecified, **/bin/sh** is used. Note that the command given with this flag can contain blank spaces if the entire command is placed within double quotes. For example:

usemouse -c "vi /etc/termcap"

is valid.

-b Suppresses bell (^G) for the duration of mouse usage. Useful with *vi*(C).

parameters
These are name=value pairs indicating what ASCII string to insert into the tty input stream, when the given event is received. Valid parameters include:

rbu=*string*	String to generate on right button up
rbd=*string*	String to generate on right button down
mbu=*string*	String to generate on middle button up
mbd=*string*	String to generate on middle button down
lbu=*string*	String to generate on left button up
lbd=*string*	String to generate on left button down
rt=*string*	String to generate on mouse right
lt=*string*	String to generate on mouse left
up=*string*	String to generate on mouse up
dn=*string*	String to generate on mouse down
ul=*string*	String to generate on mouse up-left
ur=*string*	String to generate on mouse up-right

dr=*string*	String to generate on mouse down-left
dl=*string*	String to generate on mouse down-right
hsens=*num*	Sensitivity to horizontal motion
vsens=*num*	Sensitivity to vertical motion
bells=*yes/no*	Whether to remove ^G characters

Parameters may be specified in any order. They may contain octal escapes. They may be quoted with single or double quotes if they contain blank spaces. Any parameters may be omitted and their value, if any, is taken from the configuration file.

The usemouse(C) Command

To start using the mouse with a text program, enter the command:

usemouse

This command sets the mouse for use with the default map, which is found in **/etc/default/mouse**. Alternate map files can be found in the directory **/usr/lib/mouse**. You can create your own alternate map files and place them in this directory or in your own custom map file directory. The default map file has the following values:

Mouse	Keystroke
Left Button	*vi* top of file (1G) command
Middle Button	*vi* delete character (x) command
Right Button	*vi* bottom of file (G) command
Up	Up Arrow Key
Down	Down Arrow Key
Left	Left Arrow Key
Right	Right Arrow Key
Up and Left	not defined
Up and Right	not defined
Down and Left	not defined
Down and Right	not defined
Bells	no

Invoking the *usemouse* command without specifying any options makes the mouse ready for use with a wide variety of programs or applications. Invoking *usemouse* with no options causes the mouse to use the default keystroke map. Invoking the mouse in this way creates a new command shell. You can continue to use the mouse for the duration of the shell. To terminate **usemouse**, simply enter Ctrl-D.

You can also invoke *usemouse* for the duration of a specific command:

usemouse -c *command*

This puts you in the program specified by *command* using the mouse. When you leave the program, mouse input is terminated.

Using the Mouse with Specific Programs

You can use any of several predefined maps that are set up specifically for use with different programs. (These maps are found in **/usr/lib/mouse.**) For example:

usemouse -t vi

This invokes the *vi*-specific map, which includes mapping the traditional **h-j-k-l** direction keys to the mouse movements. The terminal bell is automatically silenced by the *vi* map entry **bells=no.** This is done to prevent the bell being activated continuously when the user generates a spurious command with the mouse. (There is also a **-b** option that can be used on the *usemouse* command line to do the same thing.)

You can combine a command with a selected map file by putting both on the command line. For example:

usemouse -t vi -c vi *filename*

This invokes the *vi* map along with the command; when you quit out of *vi* the mouse disengages.

Setting Up Abbreviated (Aliased) Mouse Commands

If you plan to use the mouse frequently, you can substitute short, easy to use commands that will call up the longer command lines. This is known as command aliasing. For more information on command aliasing, consult the section "Using Aliases" in the "C-Shell" chapter of the *User's Guide.*

Specifying Map keystrokes on the Command Line

You can also specify the characters to be generated by mouse motions on the *usemouse* command line. You can specify button actions or motion actions to supplement or replace a definition from a map file. For example, assume you want to use the default *usemouse* file, but you want to redefine the middle mouse button **mbd** (middle button down) as the *vi* "i" (insert) instead of the "x" (delete character) command. The following command line does this:

usemouse -c vi mbd=i

The mouse operations are defined by a series of acronyms that are the same as used in the actual map file:

Parameter	Mouse Operation	Default
rbu	right button up	not used
rbd	right button down	1G
mbu	middle button up	not used
mbd	middle button down	x
lbu	left button up	not used
lbd	left button down	G
ul	mouse up-left	\033[A\033[C
ur	mouse up-right	\033[A\033[D
dr	mouse down-left	\033[B\033[C
dl	mouse down-right	\033[B\033[D
rt	mouse right	\033[C
lt	mouse left	\033[D
up	mouse up	\033[A
dn	mouse down	\033[B
hsens	horiz. sensitivity	5
vsens	vert. sensitivity	5

Creating Customized Maps

You can create your own personal map files for use with the mouse. The easiest way to do this is to copy the default map in **/etc/default/usemouse** and edit it. You can use quoted strings or the octal sequences found in the *ascii*(M) page. The mouse direction/button parameters are defined in the *usemouse* table above. For example, after placing a customized file, **mine**, in your home directory, you would invoke the following command to use it with the program *prog*:

usemouse -f mine -c prog

How usemouse Works

usemouse merges data from a mouse into the input stream of a tty. The mouse data is translated to arrow keys or any other arbitrary ASCII strings. Mouse movements up, down, left right, up-left, up-right, down-left, and down-right, as well as individual up and down button transitions, are programmable. This permits the mouse to be used with programs that are not designed to accept mouse input.

By default, the *usemouse* utility gets value configurations from the file **/etc/default/usemouse**.

After running the utility, provided a mouse is available, the user will be running a command with mouse motions and button events translated to ASCII strings and merged into their tty input stream. By default, the command is a shell.

Files

/dev/mouse	Directory for mouse-related special device files.
/dev/mouse/bus[0-1]	Bus mouse device files.
/dev/mouse/vpix[0-1]	vpix-mouse device files.
/dev/mouse/microsoft_ser	Microsoft serial mouse device files.
/dev/mouse/logitech_ser	Logitech serial mouse device files.
/dev/mouse/mousesys_ser	Mousesys serial mouse device files.
/dev/mouse/ttyp[0-7]	Special pseudo-tty files for mouse input.
/dev/mouse/ptyp[0-7]	Special pseudo-tty files for mouse input.
/etc/default/usemouse	Default map file for mouse-generated characters.
/usr/lib/event/devices	File containing device information for mice.
/usr/lib/event/ttys	File listing ttys eligible to use mice.
/usr/lib/mouse/*	Alternate map files for mice.

See Also

mouse(HW)

uucp, uulog, uuname

UNIX-to-UNIX system copy

Syntax

uucp [options] source-files destination-file
uulog [options] **-s**system
uulog [options] system
uulog [options] **-f**system
uuname [**-l**] [**-c**]

Description

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names that *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, to the destination. See Warnings and Notes below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information (see Warnings below for restrictions).

The shell metacharacters **?**, ***** and **[...]** appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by ~*/destination* where *destination* is appended to **/usr/spool/uucppublic**; (NOTE: This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ~/dan/ as the destination will

make the directory /usr/spool/uucppublic/dan if it does not exist and put the requested file(s) in that directory).

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(S)).

The following options are interpreted by *uucp*:

-c Do not copy local file to the spool directory for transfer to the remote machine (default).

-C Force the copy of local files to the spool directory for transfer.

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-g*grade* *Grade* is a single letter/number; lower ascii sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

-m Send mail to the requester when the copy is completed.

-n*user* Notify *user* on the remote system that a file was sent.

-r Do not start the file transfer, just queue the job.

-s*file* Report status of the transfer to *file*. Note that the *file* must be a full path name.

-x*debug_level*
Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

uulog queries a log file of *uucp* or *uuxqt* transactions in a file **/usr/spool/uucp/.Log/uucico/***system*, or **/usr/spool/uucp/.Log/uuxqt/***system*.

The options cause *uulog* to print logging information:

-s*sys*	Print information about file transfer work involving system *sys*.
-f*system*	Does a "tail -f" of the file transfer log for *system*. (You must press DELETE or BREAK to exit this function.) Other options used in conjunction with the above:
-x	Look in the *uuxqt* log file for the given system, instead of the *uucico* log file (default).
-*number*	Indicates that a "tail" command of *number* lines should be executed.

uuname lists the names of systems known to *uucp*. The **-c** option returns the names of systems known to *cu*. (The two lists are the same, unless your machine is using different *Systems* files for *cu* and *uucp*. See the *Sysfiles* file.) The **-l** option returns the local system name.

Files

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending (**/usr/spool/uucppublic**)
/usr/lib/uucp/*	other data and program files

See Also

mail(C), uustat(C), uux(C), uuxqt(C), chmod(S)

Warnings

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~/).

All files received by *uucp* will be owned by *uucp*. The **-m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **? * [...]** will not activate the **-m** option.

The forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

Notes

Protected files and files that are in protected directories that are owned by the requester can be sent by *uucp*. However, if the requester is root, and the directory is not searchable by "other" or the file is not readable by "other," the request will fail.

Standards Conformance

uucp, *uulog* and *uuname* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

uuencode, uudecode

encode/decode a binary file for transmission via mail

Syntax

uuencode [source] remotedest | **mail** sys1!sys2!..!decode
uudecode [file]

Description

uuencode and *uudecode* are used to send a binary file via uucp (or other) mail. This combination can be used over indirect mail links.

uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user ''decode'' should be filtered through the uudecode program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using *sendmail* or by making *rmail* be a link to *mail* instead of *mail* . In each case, an alias must be created in a master file to get the automatic invocation of uudecode.

If these facilities are not available, the file can be sent to a user on the remote machine who can uudecode it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

See Also

uucp(C), uux(ADM), mail(C)

Restrictions

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking *uudecode* (often *uucp)* must have write permission on the specified file.

uustat

uucp status inquiry and job control

Syntax

uustat [**-a**]
uustat [**-m**]
uustat [**-p**]
uustat [**-q**]
uustat [**-k**jobid]
uustat [**-r**jobid]
uustat [**-s**system] [**-u**user]

Description

uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

-a Output all jobs in queue.

-m Report the status of accessibility of all machines.

-p Execute a "ps -flp" for all the process-ids that are in the lock files.

-q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of **C** or **X** files, it is the age in days of the oldest **C./X.** file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the **-q** option:

```
eagle     3C    04/07-11:07NO DEVICES AVAILABLE
mh3bs3    2C    07/07-10:42SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

-k*jobid* Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

-r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the daemon.

Either or both of the following options can be specified with *uustat*:

-s*sys* Report the status of all *uucp* requests for remote system *sys*.

-u*user* Report the status of all *uucp* requests issued by *user*.

Output for both the -s and -u options has the following format:

```
eaglen0000      4/07-11:01:03(POLL)
eagleN1bd7      4/07-11:07 Seagledan522 /usr/dan/A
eagleC1bd8      4/07-11:07 Seagledan59 D.3b2al2ce4924
                4/07-11:07 Seagledanrmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).
When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

Files

/usr/spool/uucp/* spool directories

See Also

uucp(C)

Standards Conformance

uustat is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

uuto, uupick

public UNIX-to-UNIX system file copy

Syntax

uuto [options] source-files destination
uupick [**-s** system]

Description

uuto sends *source-files* to *destination*. *uuto* uses the *uucp*(C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *User* is the login name of someone on the specified system.

Two *options* are available:

-p Copy the source file into the spool directory before transmission.
-m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. By default this directory is /usr/spool/uucppublic. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(C) of the arrival of files.

uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:
from *system*: [file *file-name*] [dir *dirname*] **?**

uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

uupick invoked with the -s*system* option will only search the PUBDIR for files sent from *system*.

Files

PUBDIR /usr/spool/uucppublic public directory

See Also

mail(C), uucp(C), uustat(C), uux(C), uuclean(ADM)

Warnings

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

Standards Conformance

uupick and *uuto* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

uux

UNIX-to-UNIX system command execution

Syntax

uux [options] command-string

Description

uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

NOTE: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permitting only the receipt of mail (see *mail*(C)). (Remote execution permissions are defined in **/usr/lib/uucp/Permissions.**)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

As an example, the command

uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"

will get the *file1* and *file2* files from the ''usg'' and ''pwba'' machines, execute a *diff*(C) command and put the results in *file.diff* in the local PUBDIR/dan/ directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

uux a!cut -f1 b!/usr/file \(c!/usr/file\)

gets /usr/file from system "b" and sends it to system "a", performs a *cut* command on that file and sends the result of the *cut* command to system "c".

uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **-n** option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

- The standard input to *uux* is made the standard input to the *command-string*.

-a*name* Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)

-b Return whatever standard input was provided to the *uux* command if the exit status is non-zero.

-c Do not copy local file to the spool directory for transfer to the remote machine (default).

-C Force the copy of local files to the spool directory for transfer.

-g*grade* *Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.

-n Do not notify the user if the command fails.

-p Same as -: The standard input to *uux* is made the standard input to the *command-string*.

-r Do not start the file transfer, just queue the job.

-s*file* Report status of the transfer in *file*.

-x*debug_level*
Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

-z Send success notification to the user.

Files

/usr/spool/uucp/*	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

See Also

mail(C), uucp(C), uustat(C)

Warnings

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command. The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will NOT work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work. (If *diff* is a permitted command.)

Notes

Protected files and files that are in protected directories that are owned by the requester can be sent in commands using *uux*. However, if the requester is root, and the directory is not searchable by "other," the request will fail.

Standards Conformance

uux is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

vc

version control

Syntax

vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

Description

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the **-t** keyletter (see below). The default control character is colon (:), except as modified by the **-c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or fewer alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(C); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The **-a** keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

-a Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.

-t All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.

-c*char* Specifies a control character to be used in place of the colon.

-s Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, . . ., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition

.
.
.

:end

Used to skip lines of the standard input. If the condition is true, all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

```
<cond>     ::= [ "not" ] <or>
<or>       ::= <and> | <and> "|" <or>
<and>      ::= <exp> | <exp> "&" <and>
<exp>      ::= "(" <or> ")" | <value> <op> <value>
<op>       ::= "=" | "!=" | "<" | ">"
<value>    ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings

not may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > < all of equal precedence
&
|

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

::text Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **-a** keyletter.

:on

:off Turn on or off keyword replacement on all lines.

:ctl char
Change the control character to char.

:msg message
Prints the given message on the diagnostic output.

:err message
Prints the given message followed by:

ERROR: err statement on line ... (915)

on the diagnostic output. *vc* halts execution and returns an exit code of 1.

See Also

ed(C)

Diagnostics

0 - normal
1 - any error

vi, view, vedit

invokes a screen-oriented display editor

Syntax

vi [-option ...] [command ...] [filename ...]

view [-option ...] [command ...] [filename ...]

vedit [-option ...] [command ...] [filename ...]

Description

vi offers a powerful set of text editing operations based on a set of mnemonic commands. Most commands are single keystrokes that perform simple editing functions. *vi* displays a full screen "window" into the file you are editing. The contents of this window can be changed quickly and easily within *vi*. While editing, visual feedback is provided (the name vi itself is short for "visual").

The *view* command is the same as *vi* except that the read-only option (**-R**) is set automatically. The file cannot be changed with *view*.

The *vedit* command is the same as *vi* except for differences in the option settings. *vedit* uses **novice** mode, turns off the **magic** option, sets the option **report=1** and turns on the options **showmode** and **redraw**.

The **showmode** option informs the *vedit* user, in a message in the lower right hand corner of the screen, which mode is being used. For instance after the **ESC-i** command is used, the message reads "INSERT MODE".

Note that you can not set the **novice** option from within *vi* or *ex*. If you want to use the **novice** option you must use the *vedit* utility. (It is possible to set the **nonovice** option from within *vedit*.)

vi and the line editor *ex* are one and the same editor: the names *vi* and *ex* identify a particular user interface rather than any underlying functional difference. The differences in user interface, however, are quite striking. *ex* is a powerful line-oriented editor, similar to the editor *ed*. However, in both *ex* and *ed*, visual updating of the terminal screen is limited, and commands are entered on a command line. *vi*, on the other hand, is a screen-oriented editor designed so that what you see on the screen corresponds exactly and immediately to the contents of the file you are editing. In the following discussion, *vi* commands and options are printed in boldface type.

Options available on the *vi* command line include:

-x Encryption option; when used, the file will be encrypted as it is being written and will require an encryption key to be read. *vi* makes an educated guess to determine if a file is encrypted or not. See *crypt*(C). Also, see the **Notes** section at the end of this manual page.

-C Encryption option; the same as **-x** except that *vi* assumes files are encrypted.

-c *command*
Begin editing by executing the specified editor *command* (usually a search or positioning command).

-t *tag* Equivalent to an initial *tag* command; edits the file containin. *tag* and positions the editor at its definition.

-r *file* Used in recovering after an editor or system crash, retrieves the last saved version of the named file.

-l Specific to editing LISP, this option sets the **showmatch** and **lisp** options.

-L List the names of all files saved as a result of an editor or system crash. Files may be recovered with the **-r** option.

-w*n* Sets the default window size to *n*. Useful on dialups to start in small windows.

-R Sets a read-only option so that files can be viewed but not edited.

The Editing Buffer

vi performs no editing operations on the file that you name during invocation. Instead, it works on a copy of the file in an ''editing buffer.''

When you invoke *vi* with a single filename argument, the named file is copied to a temporary editing buffer. The editor remembers the name of the file specified at invocation, so that it can later copy the editing buffer back to the named file. The contents of the named file are not affected until the changes are copied back to the original file.

Modes of Operation

Within *vi* there are three distinct modes of operation:

Command Mode — Within command mode, signals from the keyboard are interpreted as editing commands.

Insert Mode — Insert mode can be entered by typing any of the *vi* insert, append, open, substitute, change, or replace commands. Once in insert mode, letters typed at the keyboard are inserted into the editing buffer.

ex Escape Mode — The *vi* and *ex* editors are one and the same editor differing mainly in their user interface. In *vi* , commands are usually single keystrokes. In *ex*, commands are lines of text terminated by a RETURN. *vi* has a special "escape" command that gives access to many of these line-oriented *ex* commands. To use the *ex* escape mode, type a colon (:). The colon is echoed on the status line as a prompt for the *ex* command. An executing command can be aborted by pressing INTERRUPT. Most file manipulation commands are executed in *ex* escape mode (for example, the commands to read in a file and to write out the editing buffer to a file).

Special Keys

There are several special keys in *vi*. The following keys are used to edit, delimit, or abort commands and command lines.

ESC — Used to return to *vi* command mode or to cancel partially formed commands.

RETURN — Terminates *ex* commands when in *ex* escape mode. Also used to start a newline when in insert mode.

INTERRUPT — Often the same as the DEL or RUBOUT key on many terminals. Generates an interrupt, telling the editor to stop what it is doing. Used to abort any command that is executing.

/ — Used to specify a string to be searched for. The slash appears on the status line as a prompt for a search string. The question mark (?) works exactly like the slash key, except that it is used to search backward in a file instead of forward.

:	The colon is a prompt for an *ex* command. You can then type in any *ex* command, followed by an ESC or RETURN, and the given *ex* command is executed.

The following characters are special in insert mode:

BKSP	Backs up the cursor one character on the current line. The last character typed before the BKSP is removed from the input buffer, but remains displayed on the screen.
Ctrl-U	Moves the cursor back to the first character of the insertion and restarts insertion.
Ctrl-V	Removes the special significance of the next typed character. Use Ctrl-V to insert control characters. Linefeed and Ctrl-J cannot be inserted in the text except as newline characters. Ctrl-Q and Ctrl-S are trapped by the operating system before they are interpreted by *vi*, so they too cannot be inserted as text.
Ctrl-W	Moves the cursor back to the first character of the last inserted word.
Ctrl-T	During an insertion, with the **autoindent** option set and at the beginning of the current line, entering this character will insert *shiftwidth* whitespace.
Ctrl-@	If entered as the first character of an insertion, it is replaced with the last text inserted, and the insertion terminates. Only 128 characters are saved from the last insertion. If more than 128 characters were inserted, then this command inserts no characters. A Ctrl-@ cannot be part of a file, even if quoted.

Starting and Exiting vi

To enter *vi*, enter:

vi	*Edits empty editing buffer*
vi file	*Edits named file*
vi +123 file	*Goes to line 123*
vi +45 file	*Goes to line 45*
vi +/word file	*Finds first occurrence of "word"*
vi +/tty file	*Finds first occurrence of "tty"*

There are several ways to exit the editor:

ZZ The editing buffer is written to the file *only* if any changes were made.

:x The editing buffer is written to the file *only* if any changes were made.

:q! Cancels an editing session. The exclamation mark (!) tells *vi* to quit unconditionally. In this case, the editing buffer is not written out.

vi Commands

vi is a visual editor with a window on the file. What you see on the screen is *vi*'s notion of what the file contains. Commands do not cause any change to the screen until the complete command is entered. Most commands may take a preceding count that specifies repetition of the command. This count parameter is not given in the following command descriptions, but is implied unless overridden by some other prefix argument. When *vi* gets an improperly formatted command, it rings a bell.

Cursor Movement

The cursor movement keys allow you to move your cursor around in a file. Note in particular the direction keys (if available on your terminal), the H, J, K, and L cursor keys, and SPACEBAR, BKSP, Ctrl-N, and Ctrl-P. These three sets of keys perform identical functions.

Forward Space - l, SPACEBAR, or right direction key

Syntax: **l**
SPACEBAR
right direction key

Function: Moves the cursor forward one character. If a count is given, move forward count characters. You cannot move past the end of the line.

Backspace - h, BKSP, or left direction key

Syntax: **h**
BKSP
left direction key

Function: Moves cursor backward one character. If a count is given, moves backward *count* characters. Note that you cannot move past the beginning of the current line.

Next Line - +, RETURN, j, Ctrl-N, and LF

Syntax: **+**
RETURN

Function: Moves the cursor down to the beginning of the next line.

Syntax: **j**
Ctrl-N
LF
down direction key

Function: Moves the cursor down one line, remaining in the same column. Note the difference between these commands and the preceding set of next line commands which move to the *beginning* of the next line.

Previous Line - k, Ctrl-P, and up direction key

Syntax: **k**
Ctrl-P
up direction key

Function: Moves the cursor up one line, remaining in the same column. If a count is given, the cursor is moved *count* lines.

Syntax: **-**

Function: Moves the cursor up to the beginning of the previous line. If a count is given, the cursor is moved up a *count* lines.

Beginning of Line - 0 and ^

Syntax: **^**
0

Function: Moves the cursor to the beginning of the current line. Note that **0** always moves the cursor to the first character of the current line. The caret (^) works somewhat differently: it moves to the first character on a line that is not a tab or a space. This is useful when editing files that have a great deal of indentation, such as program texts.

End of Line - $

Syntax: **$**

Function: Moves the cursor to the end of the current line. Note that the cursor resides on top of the last character on the line. If a count is given, the cursor is moved forward *count*-1 lines to the end of the line.

Goto Line - G

Syntax: [*linenumber*]**G**

Function: Moves the cursor to the beginning of the line specified by *linenumber*. If no *linenumber* is given, the cursor moves to the beginning of the *last* line in the file. To find the line number of the current line, use Ctrl-G.

Column - |

Syntax: [*column*]**|**

Function: Moves the cursor to the column in the current line given by *column*. If no *column* is given, the cursor is moved to the first column in the current line.

Word Forward - w and W

Syntax: **w**
W

Function: Moves the cursor forward to the beginning of the next word. The lowercase **w** command searches for a word defined as a string of alphanumeric characters separated by punctuation or whitespace (i.e., tab, newline, or space characters). The uppercase **W** command searches for a word defined as a string of nonwhitespace characters.

Back Word - b and B

Syntax: **b**
B

Function: Moves the cursor backward to the beginning of a word. The lowercase **b** command searches backward for a word defined as a string of alphanumeric characters separated by punctuation or whitespace (i.e., tab, newline, or space characters). The uppercase **B** command searches for a word defined as a string of non-whitespace characters. If the cursor is already within a word, it moves backward to the beginning of that word.

End - e and E

Syntax: **e**
E

Function: Moves the cursor to the end of a word. The lowercase e command moves the cursor to the last character of a word, where a word is defined as a string of alphanumeric characters separated by punctuation or whitespace (i.e., tab, newline, or space characters). The uppercase E moves the cursor to the last character of a word where a word is defined as a string of nonwhitespace characters. If the cursor is already within a word, it moves to the end of that word.

Sentence - (and)

Syntax: (
)

Function: Moves the cursor to the beginning (left parenthesis) or end of a sentence (right parenthesis). A sentence is defined as a sequence of characters ending with a period (.), question mark (?), or exclamation mark (!), followed by either two spaces or a newline. A sentence begins on the first nonwhitespace character following a preceding sentence. Sentences are also delimited by paragraph and section delimiters. See below.

Paragraph - { and }

Syntax: **}**
{

Function: Moves the cursor to the beginning ({) or end (}) of a paragraph. A paragraph is defined with the *paragraphs* option. By default, paragraphs are delimited by the nroff macros ".IP", ".LP", ".P", ".QP", and ".bp". Paragraphs also begin after empty lines.

Section - [[and]]

Syntax:]]
[[

Function: Moves the cursor to the beginning ([[) or end (]]) of a section. A section is defined with the *sections* option. By default, sections are delimited by the nroff macros ".NH" and ".SH". Sections also start at formfeeds (Ctrl-L) and at lines beginning with a brace ({).

Match Delimiter - %

Syntax: %

Function: Moves the cursor to a matching delimiter, where a delimiter is a parenthesis, a bracket, or a brace. This is useful when matching pairs of nested parentheses, brackets, and braces.

Home - H

Syntax: [*offset*]**H**

Function: Moves the cursor to the upper left corner of the screen. Use this command to quickly move to the top of the screen. If an *offset* is given, the cursor is homed *offset*-1 number of lines from the top of the screen. Note that the command "dH" deletes all lines from the current line to the top line shown on the screen.

Middle Screen - M

Syntax: **M**

Function: Moves the cursor to the beginning of the screen's middle line. Use this command to quickly move to the middle of the screen from either the top or the bottom. Note that the command "dM" deletes from the current line to the line specified by the **M** command.

Lower Screen - L

Syntax: [*offset*]L

Function: Moves the cursor to the lowest line on the screen. Use this command to quickly move to the bottom of the screen. If an *offset* is given, the cursor is homed *offset*-1 number of lines from the bottom of the screen. Note that the command "dL" deletes all lines from the current line to the bottom line shown on the screen.

Previous Context - `` and ''

Syntax: ''
'*character*
``
`*character*

Function: Moves the cursor to previous context or to context marked with the **m** command. If the single quotation mark or back quotation mark is doubled, the cursor is moved to previous context. If a single character is given after either quotation mark, the cursor is moved to the location of the specified mark as defined by the **m** command. Previous context is the location in the file of the last "nonrelative" cursor movement. The single quotation mark (´) syntax is used to move to the beginning of the line representing the previous context. The back quotation mark (`) syntax is used to move to the previous context *within* a line.

The Screen Commands

The screen commands are *not* cursor movement commands and cannot be used in delete commands as the delimiters of text objects. However, the screen commands do move the cursor and are useful in paging or scrolling through a file. These commands are described below:

Scroll - Ctrl-U and Ctrl-D

Syntax: [*size*]**Ctrl-U**
[*size*]**Ctrl-D**

Function: Scrolls the screen up a half window (Ctrl-U) or down a half window (Ctrl-D). If *size* is given, the scroll is *size* number of lines. This value is remembered for all later scrolling commands.

Page - Ctrl-F and Ctrl-B

Syntax: **Ctrl-F**
Ctrl-B

Function: Pages screen forward and backward. Two lines of continuity are kept between pages if possible. A preceding count gives the number of pages to move forward or backward.

Status - Ctrl-G

Syntax: **BELL**
Ctrl-G

Function: Displays *vi* status on status line. This gives you the name of the file you are editing, whether it has been modified, the current line number, the number of lines in the file, and the percentage of the file (in lines) that precedes the cursor.

Zero Screen - z

Syntax: [*linenumber*]**z**[*size*]**RETURN**
[*linenumber*]**z**[*size*]**.**
[*linenumber*]**z**[*size*]**-**

Function: Redraws the display with the current line placed at or "zeroed" at the top, middle, or bottom of the screen, respectively. If you give a *size*, the number of lines displayed is equal to *size*. If a preceding *linenumber* is given, the given line is placed at the top of the screen. If the last argument is a RETURN, the current line is placed at the top of the screen. If the last argument is a period (.), the current line is placed in the middle of the screen. If the last argument is a minus sign (-), the current line is placed at the bottom of the screen.

Redraw - Ctrl-R or Ctrl-L

Syntax: **Ctrl-R**
Ctrl-L
(Command depends on terminal type.)

Function: Redraws the screen. Use this command to erase any system messages that may scramble your screen. Note that system messages do not affect the file you are editing.

Text Insertion

The text insertion commands always place you in insert mode. Exit from insert mode is always done by pressing ESC. The following insertion commands are "pure" insertion commands; no text is deleted when you use them. This differs from the text modification commands, change, replace, and substitute, which delete and then insert text in one operation.

Insert - i and I

Syntax: **i[*text*]ESC**
I[*text*]ESC

Function: Insert *text* in editing buffer. The lowercase **i** command places you in insert mode. *Text* is inserted *before* the character beneath the cursor. To insert a newline, press a RETURN. Exit insert mode by typing the ESC key. The uppercase **I** command places you in insert mode, but begins text insertion at the beginning of the current line, rather than before the cursor.

Append - a and A

Syntax: **a[*text*]ESC**
A[*text*]ESC

Function: Appends *text* to the editing buffer. The lowercase a command works exactly like the lowercase i command, except that text insertion begins after the cursor and not before. This is the one way to add text to the end of a line. The uppercase **A** command begins appending text at the end of the current line rather than after the cursor.

Open New Line - o and O

Syntax: **o[*text*]ESC**
O[*text*]ESC

Function: Opens a new line and inserts text. The lowercase **o** command opens a new line below the current line; uppercase **O** opens a new line *above* the current line. After the new line has been opened, both these commands work like the **I** command.

Text Deletion

Many of the text deletion commands use the D key as an operator. This operator deletes text objects delimited by the cursor and a cursor movement command. Deleted text is always saved away in a buffer. The delete commands are described below:

Delete Character - x and X

Syntax: **x**
X

Function: Deletes a character. The lowercase **x** command deletes the character beneath the cursor. With a preceding count, *count* characters are deleted to the right beginning with the character beneath the cursor. This is a quick and easy way to delete a few characters. The uppercase X command deletes the character just before the cursor. With a preceding count, *count* characters are deleted backward, beginning with the character just before the cursor.

Delete - d and D

Syntax: **d***cursor-movement*
dd
D

Function: Deletes a text object. The lowercase **d** command takes a *cursor-movement* as an argument. If the *cursor-movement* is an intraline command, deletion takes place from the cursor to the end of the text object delimited by the *cursor-movement*. Deletion forward deletes the character beneath the cursor; deletion backward does not. If the *cursor-movement* is a multi-line command, deletion takes place from and including the current line to the text object delimited by the *cursor-movement*.

The **dd** command deletes whole lines. The uppercase **D** command deletes from and including the cursor to the end of the current line.

Deleted text is automatically pushed on a stack of buffers numbered 1 through 9. The most recently deleted text is also placed in a special delete buffer that is logically buffer 0. This special buffer is the default buffer for all (put) commands using the double quotation mark (") to specify the number of the buffer for delete, put, and yank commands. The buffers 1 through 9 can be accessed with the **p** and **P** (put) commands by appending the double quotation mark (") to the number of the buffer. For example:

"4p

puts the contents of delete buffer number 4 in your editing buffer just below the current line. Note that the last deleted text is "put" by default and does not need a preceding buffer number.

Text Modification

The text modification commands all involve the replacement of text with other text. This means that some text will necessarily be deleted. All text modification commands can be "undone" with the **u** command:

Undo - u and U

Syntax: **u**
U

Function: Undoes the last insert or delete command. The lowercase **u** command undoes the last insert or delete command. This means that after an insert, **u** deletes text; and after a delete, **u** inserts text. For the purposes of undo, all text modification commands are considered insertions.

The uppercase **U** command restores the current line to its state before it was edited, no matter how many times the current line has been edited since you moved to it.

Repeat - .

Syntax: **.**

Function: Repeats the last insert or delete command. A special case exists for repeating the **p** and **P** "put" commands. When these commands are preceded by the name of a delete buffer, successive **u** commands display the contents of the delete buffers.

Change - c and C

Syntax: **c***cursor-movement text* **ESC**
C*text* **ESC**
cc*text* **ESC**

Function: Changes a text object and replaces it with *text* . Text is inserted as with the i command. A dollar sign ($) marks the extent of the change. The **c** command changes arbitrary text objects delimited by the cursor and a *cursor-movement* . The **C** and **cc** commands affect whole lines and are identical in function.

Replace - r and R

Syntax: **r***char*
R*text* **ESC**

Function: Overstrikes character or line with *char* or *text* , respectively. Use **r** to overstrike a single character and **R** to overstrike a whole line. A count multiplies the replacement text count times.

Substitute - s and S

Syntax: s*text* **ESC**
S*text* **ESC**

Function: Substitutes current character or current line with *text.* Use s to replace a single character with new text. Use S to replace the current line with new text. If a preceding count is given, *text* substitutes for count number of characters or lines depending on whether the command is s or S, respectively.

Filter - !

Syntax: *!cursor-movement cmd* **RETURN**

Function: Filters the text object delimited by the cursor and *cursor-movement* through the UNIX command, *cmd*. For example, the following command sorts all lines between the cursor and the bottom of the screen, substituting the designated lines with the sorted lines:

!Lsort

Arguments and shell metacharacters may be included as part of *cmd*; however, standard input and output are always associated with the text object being filtered.

Join Lines - J

Syntax: **J**

Function: Joins the current line with the following line. If a *count* is given, *count* lines are joined.

Shift - < and >

Syntax: >[*cursor-movement*]
<[*cursor-movement*]
>>
<<

Function: Shifts text right (>) or left (<). Text is shifted by the value of the option *shiftwidth*, which is normally set to eight spaces. Both the > and < commands shift all lines in the text object delimited by the current line and *cursor-movement*. The >> and << commands affect whole lines. All versions of the command can take a preceding count that acts to multiply the number of objects affected.

Text Movement

The text movement commands move text in and out of the named buffers *a-z* and out of the delete buffers *1-9*. These commands either "yank" text out of the editing buffer and into a named buffer or "put" text into the editing buffer from a named buffer or a delete buffer. By default, text is put and yanked from the "unnamed buffer", which is also where the most recently deleted text is placed. Thus it is quite reasonable to delete text, move your cursor to the location where you want the deleted text placed, and then put the text back into the editing buffer at this new location with the **p** or **P** command.

The named buffers are most useful for keeping track of several chunks of text that you want to keep on hand for later access, movement, or rearrangement. These buffers are named with the letters *a* through *z*. To refer to one of these buffers (or one of the numbered delete buffers) in a command, use a quotation mark. For example, to yank a line into the buffer named *a*, enter:

"ayy

To put this text back into the file, enter:

"ap

If you delete text in the buffer named *A* rather than *a*, text is appended to the buffer.

Note that the contents of the named buffers are not destroyed when you switch files. Therefore, you can delete or yank text into a buffer, switch files, and then do a put. Buffer contents are *destroyed* when you exit the editor, so be careful.

Put - p and P

Syntax: ["*alphanumeric*]**p**
["*alphanumeric*]**P**

Function: Puts text from a buffer into the editing buffer. If no buffer name is specified, text is put from the unnamed buffer. The lowercase **p** command puts text either below the current line or after the cursor, depending on whether the buffer contains a partial line or not. The uppercase **P** command puts text either above the current line or before the cursor, again depending on whether the buffer contains a partial line or not.

Yank - y and Y

Syntax: ["*letter*]**y***cursor-movement*
["*letter*]**yy**
["*letter*]**Y**

Function: Copies text in the editing buffer to a named buffer. If no buffer name is specified, text is yanked into the unnamed buffer. If an uppercase *letter* is used, text is appended to the buffer and does not overwrite and destroy the previous contents. When a *cursor-movement* is given as an argument, the delimited text object is yanked. The **Y** and **yy** commands yank a single line, or, if a preceding count is given, multiple lines can be yanked.

Searching

The search commands search either forward or backward in the editing buffer for text that matches a given regular expression.

Search - / and ?

Syntax: /[*pattern*]/[*offset*]**RETURN**
/[*pattern*]**RETURN**
?[*pattern*]?[*offset*]**RETURN**
?[*pattern*]**RETURN**

Function: Searches forward (/) or backward (?) for *pattern*. A string is actually a regular expression. The trailing delimiter is not required. If no *pattern* is given, then last *pattern* searched for is used. After the second delimiter, an *offset* may be given, specifying the beginning of a line relative to the line on which *pattern* was found. For example:

/word/-

finds the beginning of the line immediately preceding the line containing ''word'' and the following command:

/word/+2

finds the beginning of the line two lines after the line containing ''word''. See also the *ignorecase* and *magic* options.

Next String - n and N

Syntax: **n**
N

Function: Repeats the last search command. The **n** command repeats the search in the same direction as the last search command. The **N** command repeats the search in the opposite direction of the last search command.

Find Character - f and F

Syntax: **f***char*
F*char*
;
,

Function: Finds character *char* on the current line. The lowercase **f** searches forward on the line; the uppercase **F** searches backward. The semicolon (;) repeats the last character search. The comma (,) reverses the direction of the search.

To Character - t and T

Syntax: **t***char*
T*char*
;
,

Function: Moves the cursor up to but not on *char*. The semicolon (;) repeats the last character search. The comma (,) reverses the direction of the search.

Mark - m

Syntax: **m***letter*

Function: Marks a place in the file with a lowercase *letter*. You can move to a mark using the ''to mark'' commands described below. It is often useful to create a mark, move the cursor, and then delete from the cursor to the mark "a" with the following command:

d´a

To Mark - ´ and `

Syntax: *´letter*
`letter

Function: Move to *letter*. These commands let you move to the location of a mark. Marks are denoted by single lowercase alphabetic characters. Before you can move to a mark, it must first be created with the **m** command. The back quotation mark (`) moves you to the exact location of the mark within a line; the forward quotation mark (´) moves you to the beginning of the line containing the mark. Note that these commands are also legal cursor movement commands.

Exit and Escape Commands

There are several commands that are used to escape from *vi* command mode and to exit the editor. These are described in the following section.

ex Escape - :

Syntax: **:**

Function: Enters *ex* escape mode to execute an *ex* command. The colon appears on the status line as a prompt for an *ex* command. You then can enter an *ex* command line terminated by either a RETURN or an ESC and the *ex* command will execute. You are then prompted to type RETURN to return to *vi* command mode. During the input of the *ex* command line or during execution of the *ex* command, you may press INTERRUPT to stop what you are doing and return to *vi* command mode.

Exit Editor - ZZ

Syntax: **ZZ**

Function: Exit *vi* and write out the file if any changes have been made. This returns you to the shell from which you started *vi*.

Quit to ex - Q

Syntax: **Q**

Function: Enters the *ex* editor. When you do this, you will still be editing the same file. You can return to *vi* by entering the *vi* command from *ex*.

ex Commands

Entering the colon (:) escape command when in command mode produces a colon prompt on the status line. This prompt is for a command available in the line-oriented editor, *ex*. In general, *ex* commands let you write out or read in files, escape to the shell, or switch editing files.

Many of these commands perform actions that affect the "current" file by default. The current file is normally the file that you named when you started *vi*, although the current file can be changed with the "file" command, **f**, or with the "next" command, **n**. In most respects, these commands are identical to similar commands for the editor, *ed*. All such *ex* commands are aborted by either RETURN or ESC. We shall use RETURN in our examples. Command entry is terminated by typing INTERRUPT.

Command Structure

Most *ex* command names are English words, and initial prefixes of the words are acceptable abbreviations. In descriptions, only the abbreviation is discussed, since this is the most frequently used form of the command. The ambiguity of abbreviations is resolved in favor of the more commonly used commands. As an example, the command **substitute** can be abbreviated **s** , while the shortest available abbreviation for the **set** command is **se.**

Most commands accept prefix addresses specifying the lines in the file that they are to affect. A number of commands also may take a trailing *count* specifying the number of lines to be involved in the command. Counts are rounded down if necessary. Thus, the command "10p" displays the tenth line in the buffer while "move 5" moves the current line after line 5.

Some commands take other information or parameters, stated after the command name. Examples might be option names in a set command, such as "set number", a filename in an **edit** command, a regular expression in a **substitute** command, or a target address for a **copy** command. For example:

1,5 copy 25

A number of commands have variants. The variant form of the command is invoked by placing an exclamation mark (!) immediately after the command name. Some of the default variants may be controlled by options; in this case, the exclamation mark turns off the meaning of the default.

In addition, many commands take flags, including the characters "p" and "l". A "p" or "l" must be preceded by a blank or tab. In this case, the command abbreviated by these characters is executed after the command completes. Since *ex* normally displays the new current line after each change, **p** is rarely necessary. Any number of plus (+) or minus (-) characters may also be given with these flags. If they appear, the specified offset is applied to the current line value before the printing command is executed.

Most commands that change the contents of the editor buffer give feedback if the scope of the change exceeds a threshold given by the **report option.** This feedback helps to detect undesirably large changes so that they may be quickly and easily reversed with the **undo** command. After commands with global effect, you will be informed if the net change in the number of lines in the buffer during this command exceeds this threshold.

Command Addressing

The following specifies the line addressing syntax for *ex* commands:

.
: The current line. Most commands leave the current line as the last line which they affect. The default address for most commands is the current line, thus "." is rarely used alone as an address.

n
: The *n*th line in the editor's buffer, lines being numbered sequentially from 1.

$
: The last line in the buffer.

%
: An abbreviation for "1,$", the entire buffer.

+*n* or -*n*
: An offset, *n* relative to the current buffer line. The forms ".+3" "+3" and "+++" are all equivalent. If the current line is line 100 they all address line 103.

/pattern/ or *?pattern?*
: Scan forward and backward respectively for a text matching the regular expression given by *pattern*. Scans normally wrap around the end of the buffer. If all that is desired is to print the next line containing *pattern*, the trailing slash (/) or question mark (?) may be omitted. If *pattern* is omitted or explic-

itly empty, the string matching the last specified regular expression is located. The forms "RETURN" and "?RETURN" scan using the last named regular expression. After a substitute, "RETURN" and "??RETURN" would scan using that substitute's regular expression.

´´ or *´x*

Before each nonrelative motion of the current line dot (.), the previous current line is marked with a label, subsequently referred to with two single quotation marks (´´). This makes it easy to refer or return to this previous context. Marks are established with the *vi* **m** command, using a single lowercase letter as the name of the mark. Marked lines are later referred to with the following notation:

´x.

where *x* is the name of a mark.

Addresses to commands consist of a series of addresses, separated by a colon (,) or a semicolon (;). Such address lists are evaluated left to right. When addresses are separated by a semicolon (;) the current line (.) is set to the value of the previous addressing expression before the next address is interpreted. If more addresses are given than the command requires, all but the last one or two are ignored. If the command takes two addresses, the first addressed line must precede the second in the buffer. Null address specifications are permitted in a list of addresses, the default in this case is the current line "."; thus ",100" is equivalent to ".,100". It is an error to give a prefix address to a command which expects none.

Command Format

The following is the format for all *ex* commands:

[*address*] [*command*] [!] [*parameters*] [*count*] [*flags*]

All parts are optional depending on the particular command and its options. The following section describes specific commands.

Argument List Commands

The argument list commands allow you to work on a set of files, by remembering the list of filenames that are specified when you invoke *vi*. The **args** command lets you examine this list of filenames. The **file** command gives you information about the current file. The **n** (next) command lets you either edit the next file
in the argument list or change the list. And the **rewind** command lets

you restart editing the files in the list. All of these commands are described below:

args — The members of the argument list are displayed, with the current argument delimited by brackets. For example, a list might look like this:

file1 file2 [file3] file4 file5

The current file is *file3*.

f — Displays the current filename, whether it has been modified since the last **write** command, whether it is read-only, the current linenumber, the number of lines in the buffer, and the percentage of the buffer that you have edited. In the rare case that the current file is "[Not edited]", this is noted also; in this case you have to use **w!** to write to the file, since the editor is not sure that a **w** command will not destroy a file unrelated to the current contents of the buffer.

f *file* — The current filename is changed to *file* which is considered "[Not edited]".

n — The next file in the command line argument list is edited.

n! — This variant suppresses warnings about the modifications to the buffer not having been written out, discarding irretrievably any changes that may have been made.

n [+*command*] *filelist* — The specified *filelist* is expanded and the resulting list replaces the current argument list; the first file in the new list is then edited. If *command* is given (it must contain no spaces), then it is executed after editing the first such file.

rew — The argument list is rewound, and the first file in the list is edited.

rew! — Rewinds the argument list discarding any changes made to the current buffer.

If you use C-Shell and set the **prompt** variable to output a prompt for non-interactive shells, the prompt is interpreted as a filename when you use these commands. This causes unexpected problems. To avoid these problems, use the default **prompt** value as specified in */usr/lib/mkuser/mkuser.cshrc*.

Edit Commands

To edit a file other than the one you are currently editing, you will often use one of the variations of the **e** command.

In the following discussions, note that the name of the current file is always remembered by vi and is specified by a percent sign (%). The name of the *previous* file in the editing buffer is specified by a number sign (#).

The edit commands are described below:

e *file* Used to begin an editing session on a new file. The editor first checks to see if the buffer has been modified since the last **w** command was issued. If it has been, a warning is issued and the command is aborted. The command otherwise deletes the entire contents of the editor buffer, makes the named file the current file, and displays the new filename. After ensuring that this file is sensible, (i.e., that it is not a binary file, directory, or a device), the editor reads the file into its buffer. If the read of the file completes without error, the number of lines and characters read is displayed on the status line. If there were any non-ASCII characters in the file, they are stripped of their non-ASCII high bits, and any null characters in the file are discarded. If none of these errors occurred, the file is considered edited. If the last line of the input file is missing the trailing newline character, it is supplied and a complaint issued. The current line is initially the first line of the file.

e! *file* This variant form suppresses the complaint about modifications having been made and not written from the editor buffer, thus discarding all changes that have been made before editing the new file.

e +*n file* Causes the editor to begin editing at line *n* rather than at the first line. The argument *n* may also be an editor command containing no spaces; for example, "+/pattern".

Ctrl-^ This is a shorthand equivalent for ":e #RETURN", which returns to the previous position in the last edited file. If you do not want to write the file, you should use ":e! #RETURN" instead.

Write Commands

The write commands let you write out all or part of your editing buffer to either the current file or to some other file. These commands are described below:

w *file* — Writes changes made back to *file*, displaying the number of lines and characters written. Normally, *file* is omitted and the buffer is written to the name of the current file. If *file* is specified, text is written to that file. The editor writes to a file only if it is the current file and is edited, or if the file does not exist. Otherwise, you must give the variant form w! to force the write. If the file does not exist it is created. The current filename is changed only if there is no current filename; the current line is never changed.

If an error occurs while writing the current and edited file, the editor displays:

No write since last change

even if the buffer had not previously been modified.

w>> *file* — Appends the buffer contents at the end of an existing file. Previous file contents are not destroyed.

w! *name* — Overrides the checking of the normal write command, and writes to any file that the system permits.

w !*command* — Writes the specified lines into *command*. Note the difference between

w! *file*

which overrides checks and

w !*cmd*

which writes to a command. The output of this command is displayed on the screen and not inserted in the editing buffer.

Read Commands

The read commands let you read text into your editing buffer at any location you specify. The text you read in must be at least one line long, and can be either a file or the output from a command.

r *file* — Places a copy of the text of the given file in the editing buffer after the specified line. If no file is given, the current filename is used. The current filename is not changed unless there is none, in which case the file becomes the current name. If the file buffer is empty and there is no current name, this is treated as an **e** command.

Address 0 is legal for this command and causes the file to be read at the beginning of the buffer. Statistics are given as for the **e** command when the **r** successfully terminates. After an **r** the current line is the last line read.

r !*command* Reads the output of *command* into the buffer after the specified line. A blank or tab before the exclamation mark (!) is mandatory.

Quit Commands

There are several ways to exit *vi*. Some abort the editing session, some write out the editing buffer before exiting, and some warn you if you decide to exit without writing out the buffer. All of these ways of exiting are described below:

q Exits *vi*. No automatic write of the editor buffer to a file is performed. However, *vi* displays a warning message if the file has changed since the last **w** command was issued, and does not quit. *vi* also displays a diagnostic if there are more files in the argument list left to edit. Normally, you will wish to save your changes, and you should enter a **w** command. If you wish to discard them, enter the **q!** command variant.

q! Quits from the editor, discarding changes to the buffer without complaint.

wq *name* Like a **w** and then a **q** command.

wq! *name* Overrides checking normally made before execution of the **w** command to any file. For example, if you own a file but do not have write permission turned on, the **wq!** allows you to update the file anyway.

x *name* If any changes have been made and not written, writes the buffer out and then quits. Otherwise, it just quits.

Global and Substitute Commands

The global and substitute commands allow you to perform complex changes to a file in a single command. Learning how to use these commands is a must for an experienced *vi* user.

g/*pattern*/*cmds*

The **g** command has two distinct phases. In the first phase, each line matching *pattern* in the editing buffer is marked. Next, the given command list is executed with the current line, dot (.), initially set to each marked line.

The command list consists of the remaining commands on the current input line and may continue to multiple lines by ending all but the last such line with a backslash (\). This multiple-line option will not work from within *vi*, you must switch to *ex* to do it. If *cmds* (or the trailing slash (/) delimiter) is omitted, each line matching *pattern* is displayed.

The **g** command itself may not appear in *cmds*. The options **autoprint** and **autoindent** are inhibited during a global command and the value of the **report** option is temporarily infinite, in deference to a **report** for the entire global. Finally, the context mark (´) or (`) is set to the value of the current line (.) before the global command begins and is not changed during a global command.

The following global commands, most of them substitutions, cover the most frequent uses of the global command.

g/*s1***/p** This command simply prints all lines that contain the string "s1" .

g/*s1***/s//***s2***/** This command substitutes the *first* occurrence of "s1" on all lines that contain it with the string "s2".

g/*s1***/s//***s2***/g** This command substitutes all occurrences of "s1" with the string "s2". This includes multiple occurrences of "s1" on a line.

g/*s1***/s//***s2***/gp** This command works the same as the preceding example, except that in addition, all changed lines are displayed on the screen.

g/*s1***/s//***s2***/gc** This command prompts you to confirm that you want to make each substitution of the string "s1" with the string "s2". If you enter a Y , the given substitution is made, otherwise it is not.

g/*s0***/s/***s1***/***s2***/g** This command marks all those lines that contain the string "s0", and then for those lines only, substitutes all occurrences of the string "s1" with "s2".

g!*/pattern/cmds* This variant form of **g** runs *cmds* at each line not matching *pattern*.

g/^/s// /g This command inserts blank spaces at the beginning of each line in a file.

s/*pattern*/*repl*/*options*
On each specified line, the first instance of text matching the regular expression *pattern* is replaced by the replacement text *repl*. If the **global** indicator option character **g** appears, all instances on a line are substituted. If the **confirm** indication character **c** appears, before each substitution the line to be substituted is printed on the screen with the string to be substituted marked with caret (^) characters. By entering Y , you cause the substitution to be performed; any other input causes no change to take place. After an **s** command, the current line is the last line substituted.

v/*pattern*/*cmds*
A synonym for the **global** command variant **g!**, running the specified *cmds* on each line that does not match *pattern*.

Text Movement Commands

The text movement commands are largely superseded by commands available in *vi* command mode. However, the following two commands are still quite useful:

co *addr flags*
A copy of the specified lines is placed after *addr*, which may be "0". The current line "." addresses the last line of the copy.

[*range*]**m***addr*
The **m** command moves the lines specified by *range* after the line given by *addr*. For example, **m+** swaps the current line and the following line, since the default range is just the current line. The first of the moved lines becomes the current line (dot).

Shell Escape Commands

You will often want to escape from the editor to execute normal UNIX commands. You may also want to change your working directory so that your editing can be done with respect to a different working directory. These operations are described below:

cd *directory*
The specified *directory* becomes the current directory. If no directory is specified, the current value of the *home* option is used as the target directory. After a **cd** , the current file is not considered to have been edited so that write restrictions on preexisting files still apply.

sh A new shell is created. You may invoke as many commands as you like in this shell. To return to *vi*, enter a Ctrl-D to terminate the shell.

!*command* The remainder of the line after the exclamation (!) is sent to a shell to be executed. Within the text of *command* , the characters "%" and "#" are expanded as the filenames of the current file and the last edited file and the character "!" is replaced with the text of the previous command. Thus, in particular, "!!" repeats the last such shell escape. If any such expansion is performed, the expanded line is echoed. The current line is unchanged by this command.

If there has been "[No write]" of the buffer contents since the last change to the editing buffer, a diagnostic is displayed before the command is executed as a warning. A single exclamation (!) is displayed when the command completes.

If you use C-Shell and set the **prompt** variable to output a prompt for non-interactive shells, the prompt is interpreted as an argument for *command* in shell escapes. This causes unexpected problems. To avoid these problems, use the default **prompt** value as specified in */usr/lib/mkuser/mkuser.cshrc.*

Other Commands

The following command descriptions explain how to use miscellaneous *ex* commands that do not fit into the above categories.

The **abbr**, **map**, and **set** commands can also be defined with the **EXINIT** environment variable, which is read by the editor each time it starts up. For more information, see *environ*(M). Alternatively, these commands can be placed in a **.exrc** file in your home directory, which the editor reads if **EXINIT** is not defined.

abbr Maps the first argument to the following string. For example, the following command

:abbr rainbow yellow green blue red

maps "rainbow" to "yellow green blue red". Abbreviations can be turned off with the **unabbreviate** command, as in:

:una rainbow

map, map! Maps any character or escape sequence to a command sequence. For example, the following command maps the CTRL-A key to a shell escape that runs the *clear*(C) command:

map ^A :!clear^M

To include the CTRL-A and CTRL-M characters in the mapping, you must use *vi*'s CTRL-V escape.

Characters mapped with **map** work in command mode, while characters mapped with **map!** work in insert mode. Characters mapped with **map!** cannot be unmapped using **unmap**.

nu Displays each specified line preceded by its buffer line number. The current line is left at the last line displayed. To get automatic line numbering of lines in the buffer, set the *number* option.

preserve The current editor buffer is saved as though the system had just crashed. This command is for use only in emergencies when a **w** command has resulted in an error and you do not know how to save your work.

= Displays the line number of the addressed line. The current line is unchanged.

recover *file*

Recovers *file* from the system save area. The system saves a copy of the editing buffer only if you have made changes to the file, the system crashes, or you execute a **preserve** command. When you use **preserve** , you are notified by mail when a file is saved.

set *argument*

With no arguments, **set** displays those options whose values have been changed from their defaults; with the argument **all**, it displays all of the option values.

Giving an option name followed by a question mark (?) causes the current value of that option to be displayed. The question mark is unnecessary unless the option is a Boolean value. Switch options are given values either with:

set *option*

to turn them on or:

set no*option*

to turn them off. String and numeric options are assigned with:

set *option*=value

More than one option can be given to *set*; all are interpreted from left to right. See "Options" for a complete list and descriptions.

tag *label* The focus of editing switches to the location of *label*. If necessary, *vi* will switch to a different file in the current directory to find *label*. If you have modified the current file before giving a **tag** command, you must first write it out. If you give another **tag** command with no argument, the previous *label* is used.

Similarly, if you press Ctrl-], *vi* searches for the word immediately after the cursor as a tag. This is equivalent to entering ":tag", the word following the cursor, and then pressing the RETURN key.

The tags file is normally created by a program such as **ctags,** and consists of a number of lines with three fields separated by blanks or tabs. The first field gives the name of the tag, the second the name of the file where the tag resides, and the third gives an addressing form which can be used by the editor to find the tag. This field is usually a contextual scan using */pattern/* to be immune to minor changes in the file. Such scans are always performed as if the **nomagic** option was set. The tag names in the tags file must be sorted alphabetically.

unmap Unmaps any character or escape sequence that has been mapped using the **map** command.

Options

There are a number of options that can be set to affect the *vi* environment. These can be set with the *ex* **set** command while editing, with the **EXINIT** environment variable, or in the *vi* start-up file, **.exrc**. This file normally sets the user's preferred options so that they do not need to be set manually each time you invoke *vi*.

The first thing that must be done before you can use *vi*, is to set the terminal type so that *vi* understands how to talk to the particular terminal you are using.

There are only two kinds of options: switch options and string options. A switch option is either on or off. A switch is turned off by prefixing the word *no* to the name of the switch within a **set** command. String options are strings of characters that are assigned values with the syntax *option=string*. Multiple options may be specified on a line. *vi* options are listed below:

autoindent, ai default: **noai**

Can be used to ease the preparation of structured program text. For each line created by an append, change, insert, open, or substitute operation, *vi* looks at the preceding line to determine and insert an appropriate amount of indentation. To back the cursor up to the preceding tab stop, press Ctrl-D. The tab stops going backward are defined as multiples of the **shiftwidth** option. You cannot backspace over the indent, except by pressing Ctrl-D.

Specially processed in this mode is a line with no characters added to it, which turns into a completely blank line (the whitespace provided for the **autoindent** is discarded). Also, specially processed in this mode are lines beginning with a caret (^) and immediately followed by a Ctrl-D. This causes the input to be repositioned at the beginning of the line, but retains the previous indent for the next line. Similarly, a "0" followed by a Ctrl-D, repositions the cursor at the beginning without retaining the previous indent. **Autoindent** doesn't happen in global commands.

autoprint ap default: **ap**

Causes the current line to be displayed after each *ex* **copy, move,** or **substitute** command. This has the same effect as supplying a trailing "p" to each such command. **Autoprint** is suppressed in globals, and only applies to the last command on a line.

autowrite, aw default: **noaw**

Causes the contents of the buffer to be automatically written to the current file if you have modified it when you give a **next, rewind, tag,** or **!** command, or a Ctrl-^ (switch files) or Ctrl-] (tag go to) command.

beautify, bf default: **nobeautify**

Causes all control characters except tab, newline and formfeed to be discarded from the input. A complaint is registered the first time a backspace character is discarded. **Beautify** does not apply to command input.

directory, dir default: **dir=/tmp**

Specifies the directory in which *vi* places the editing buffer file. If the directory does not have write permission, the editor will exit abruptly when it fails to write to the buffer file.

edcompatible default: **noedcompatible**

Causes the presence or absence of **g** and **c** suffixes on substitute commands to be remembered, and to be toggled on and off by repeating the suffixes. The suffix **r** causes the substitution to be like the tilde (~) command, instead of like the ampersand command (**&**).

errorbells, eb default: **noeb**

Error messages are preceded by a bell. If possible, the editor always places the error message in inverse video instead of ringing the bell.

hardtabs, ht default: **ht=8**
Gives the boundaries on which terminal hardware tabs are set or on which tabs the system expands.

ignorecase, ic default: **noic**
Maps all uppercase characters in the text to lowercase in regular expression matching. In addition, all uppercase characters in regular expressions are mapped to lowercase except in character class specifications enclosed in brackets.

lisp default: **nolisp**
Autoindent indents appropriately for LISP code, and the ( ) { } [[and]] commands are modified to have meaning for LISP.

list default: **nolist**
All printed lines are displayed, showing tabs and end-of-lines.

magic default: **magic**
If **nomagic** is set, the number of regular expression metacharacters is greatly reduced, with only up-arrow (^) and dollar sign ($) having special effects. In addition, the metacharacters "~" and "&" in replacement patterns are treated as normal characters. All the normal metacharacters may be made **magic** when **nomagic** is set by preceding them with a backslash (\).

mesg default: **nomesg**
Causes write permission to be turned off to the terminal while you are in visual mode, if **nomesg** is set. This prevents people writing to your screen with the UNIX **write** command and scrambling your screen as you edit.

number, n default: **nonumber**
Causes all output lines to be printed with their line numbers.

open default: **open**
If set to **noopen**, the commands **open** and **visual** are not permitted from *ex*. This is set to prevent confusion resulting from accidental entry to open or visual mode.

optimize, opt default: **optimize**
Output of text to the screen is expedited by setting the terminal so that it does not perform automatic carriage returns when displaying more than one line of output, thus greatly speeding output on terminals without addressable cursors when text with leading whitespace is printed.

paragraphs, para default: **para =IPLPPPQPP TPbp**
Specifies paragraph delimiters for the { and } operations. The pairs of characters in the option's value are the names of the nroff macros that start paragraphs.

prompt default: **prompt**
ex input is prompted for with a colon (:). If **noprompt** is set, when *ex* command mode is entered with the **Q** command, no colon prompt is displayed on the status line.

redraw default: **noredraw**
The editor simulates (using great amounts of output), an intelligent terminal on a dumb terminal. Useful only at very high speed.

remap default: **remap**
If on, mapped characters are repeatedly tried until they are unchanged. For example, if *o* is mapped to *O* and *O* is mapped to *I*, *o* will map to *I* if remap is set, and to *O* if **noremap** is set.

report default: **report=5**
Specifies a threshold for feedback from commands. Any command that modifies more than the specified number of lines will provide feedback as to the scope of its changes. For global commands and the undo command, the net change in the number of lines in the buffer is presented at the end of the command. Thus notification is suppressed during a **g** command on the individual commands performed.

scroll default: **scroll=½ window**
Determines the number of logical lines scrolled when Ctrl-D is received from a terminal input in command mode, and the number of lines displayed by a command mode **z** command (double the value of *scroll*).

sections default: **sections=**SHNHH HU
Specifies the section macros for the [[and]] operations. The pairs of characters in the option's value are the names of the nroff macros that start paragraphs.

shell, sh default: **sh=**/bin/sh
Gives the pathname of the shell forked for the shell escape command (!), and by the **shell** command. The default is taken from SHELL in the environment, if present.

shiftwidth, sw default:sw=8
Gives the width of a software tab stop, used in reverse tabbing with Ctrl-D when using **autoindent** to append text, and by the shift commands.

showmatch, sm default: **nosm**
When a) or } is typed, moves the cursor to the matching (or { for one second if this matching character is on the screen.

showmode default: **noshowmode**
Causes the message "INPUT MODE to appear on lower right corner of the screen when insert mode is activated.

slowopen default: **noslowopen**
Postpones update of the display during inserts.

tabstop, ts default: **ts=8**
The editor expands tabs in the input file to be on *n* boundaries for the purposes of display.

taglength, tl default: **tl=0**
The first *n* characters in a tag name are significant, but all others are ignored. A value of zero (the default) means that all characters are significant.

tags default: **tags=tags /usr/lib/tags**
A path of files to be used as tag files for the tag command. A requested tag is searched for in the specified files, sequentially. By default, files named *tag* are searched for in the current directory and in **/usr/lib**.

term default=value of shell TERM variable
The terminal type of the output device.

terse default: **noterse**
Shorter error diagnostics are produced for the experienced user.

timeout , to default: **noto**
Eliminates the 1 second time limit for **maps** (character mappings).

warn default: **warn**
Warn if there has been "[No write since last change]" before a shell escape command (**!**).

window default: **window** = speed dependent
This specifies the number of lines in a text window. The default is 8 at slow speeds (600 baud or less), 16 at medium speed (1200 baud), and the full screen (minus one line) at higher speeds.

w300, w1200, w9600
These are not true options but set **window** (above) only if the speed is slow (300), medium (1200), or high (9600), respectively.

wrapscan, ws default: **ws**
Searches, using the regular expressions in addressing, will wrap around past the end of the file.

wrapmargin, wm default: **wm=0**
Defines the margin for automatic insertion of newlines during text input. A value of zero specifies no wrap margin.

writeany, wa default: **nowa**
Inhibits the checks normally made before **write** commands, allowing a write to any file that the system protection mechanism will allow.

Regular Expressions

A regular expression specifies a set of strings of characters. A member of this set of strings is said to be "matched" by the regular expression. *vi* remembers two previous regular expressions: the previous regular expression used in a substitute command and the previous regular expression used elsewhere, referred to as the previous *scanning* regular expression. The previous regular expression can always be referred to by a null regular expression: e.g., "//" or "??".

The regular expressions allowed by *vi* are constructed in one of two ways depending on the setting of the **magic** option. The *ex* and *vi* default setting of **magic** gives quick access to a powerful set of regular expression metacharacters. The disadvantage of **magic** is that the user must remember that these metacharacters are **magic** and precede them with the backslash (\) to use them as "ordinary" characters. With **nomagic** set, regular expressions are much simpler, there being only two metacharacters. The power of the other metacharacters is still available by preceding the now ordinary character with a "\". Note that "\" is always a metacharacter. In this discussion, the magic option is assumed. With **nomagic** , the only special characters are the caret (^) at the beginning of a regular expression, the dollar sign ($) at the end of a regular expression, and the backslash (\). The tilde (~) and the ampersand (&) also lose their special meanings related to the replacement pattern of a substitute.

The following basic constructs are used to construct **magic** mode regular expressions.

char An ordinary character matches itself. Ordinary characters are any characters except a caret (^) at the beginning of a line, a dollar sign ($) at the end of line, a star (*) as any character other than the first, and any of the following characters:

. \ [~

These characters must be preceded by a backslash (\) if they are to be treated as ordinary characters.

^ At the beginning of a pattern, forces the match to succeed only at the beginning of a line.

$ At the end of a regular expression, forces the match to succeed only at the end of the line.

. Matches any single character except the newline character.

\< Forces the match to occur only at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> Similar to "\<", but matching the end of a "word", i.e., either the end of the line or before a character which is not a letter, a digit, or the underline character.

[*string*]
Matches any single character in the class defined by *string*. Most characters in *string* define themselves. A pair of characters separated by a dash (-) in *string* defines the set of characters between the specified lower and upper bounds, thus "[a-z]" as a regular expression matches any single lowercase letter. If the first character of *string* is a caret (^) then the construct matches those characters which it otherwise would not. Thus "[^a-z]" matches anything but a lowercase letter or a newline. To place any of the characters caret, left bracket, or dash in *string* they must be escaped with a preceding backslash (\).

The concatenation of two regular expressions first matches the leftmost regular expression and then the longest string that can be recognized as a regular expression. The first part of this new regular expression matches the first regular expression and the second part matches the second. Any of the single character matching regular expressions mentioned above may be followed by a star () to form a regular expression that matches zero or more adjacent occurrences of the characters matched by the prefixing regular expression. The tilde (~) may be used in a regular expression to match the text that defined the replacement part of the last s command. A regular expression may be enclosed between the sequences "\(" and "\)" to remember the text matched by the enclosed regular expression. This text can later be interpolated into the replacement text using the following notation:

digit

where *digit* enumerates the set of remembered regular expressions.

The basic metacharacters for the replacement pattern are the ampersand (&) and the tilde (~); these are given as "\&" and "\~" when **nomagic** is set. Each instance of the ampersand is replaced by the characters matched by the regular expression. In the replacement pattern, the tilde stands for the text of the previous replacement pattern.

Other metasequences possible in the replacement pattern are always introduced by a backslash (\). The sequence "*n*" is replaced by the text matched by the *n*th regular subexpression enclosed between "\(" and "\)". When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of "\(" starting from the left. The sequences "\u" and "\l" cause the immediately following character in the replacement to be converted to uppercase or lowercase, respectively, if this character is a letter. The sequences "\U" and "\L" turn such conversion on, either until "\E" or "\e" is encountered, or until the end of the replacement pattern.

Files

/tmp	default directory where temporary work files are placed; it can be changed using the **directory** option (see the *ex*(C) **set** command.).
/usr/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.COREterm/?/*	subset of compiled terminal description database

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

The **/usr/lib/ex3.7preserve** program can be used to restore *vi* buffer files that were lost as a result of a system crash. The program searches the **/tmp** directory for *vi* buffer files and places them in the directory **/usr/preserve**. The owner can retrieve these files using the **-r** option.

The **/usr/lib/ex3.7preserve** program must be placed in the system startup file, **/etc/rc.d/3/recovery**, before the command that cleans out the **/tmp** directory. See the *System Administrator's Guide* for more information on the **/etc/rc2** scripts.

Two options, although they continue to be supported, have been replaced in the documentation by the options that follow the Command Syntax Standard (see *intro*(C)). A **-r** option that is not followed with an argument has been replaced by **-L** and *+command* has been replaced by **-c** *command*.

vi does not strip the high bit from 8 bit characters read in from text files, text insertion, and editing commands. It does not look for magic numbers of object files when reading in a text file. It also writes out text and displays text without stripping the high bit.

vi uses the **LC_CTYPE** environment variable to determine if a character is printable, displaying the octal codes of non-printable 8 bit characters. It also uses **LC_CTYPE** and **LANG** to convert between upper and lowercase characters for the tilde command and for the **ignorecase** option.

When the percent sign (%) is used in a shell escape from *vi* via the exclamation mark (!) the % is replaced with the name of the file being edited. In previous versions of *vi*, each character in this replacement had the high bit set to 1 to quote it; in the current version of *vi* it is left alone.

Warnings

Tampering with the entries in **/usr/lib/.COREterm/?/*** or **/usr/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs such as *vi* that expect all entries to be present and correct. In particular, removing the "dumb" terminal entry may cause unexpected problems.

Software tabs using **^T** work only immediately after the **autoindent**.

Left and right shifts on intelligent terminals do not make use of insert and delete operations in the terminal.

Refer to the *crypt*(C) page for information about restrictions on the availability of encryption options.

Standards Conformance

vedit and *view* are conformant with:

AT&T SVID Issue 2, Select Code 307-127.

vi is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

vidi

sets the font and video mode for a video device

Syntax

vidi [**-d**] [**-f** fontfile] command

Description

vidi sets video mode or loads/extracts a font from the device that is the current standard input; without arguments, it lists the all of the valid video mode and font commands.

Some video cards support changeable character fonts. The *vidi* font commands (font8x8, font8x14, and font8x16) are used to load and extract fonts from the tables stored in the kernel. If neither of the **-d** or **-f** options has been specified, *vidi* will attempt to load the specified font from **i/usr/lib/vidi/***fontname.* he **-d** option causes *vidi* to read the specified font from the kernel and write (dump) the font to the standard output.

The **-f** option is used to load fonts other than those in /usr/lib/vidi or to specify an output file other than standard output. When loading a font, **-f** *filename* will load the font from *filename* instead of from /usr/lib/vidi/*fontname*. When extracting a font (**-d** option) **-f** *filename* causes *vidi* to write the extracted font into *filename* instead of writing the font to the standard output.

The other *vidi* commands set the video mode of the video adapter connected to *vidi*'s standard input. The commands are :

mono
: move current screen to the monochrome adapter.

cga
: move current screen to the Color Graphics adapter.

ega
: move current screen to the Enhanced Graphics adapter.

vga
: move current screen to the Video Graphics adapter.

internal
: activate the internal monitor on portable with a plasma screen.

external
: activate the external monitor on portable with a plasma screen.

Text Modes				
Command	Cols	Rows	Font	Adapter
c40x25	40	25	8x8	CGA (EGA VGA)
e40x25	40	25	8x14	EGA (VGA)
v40x25	40	25	8x16	VGA
m80x25	80	25	8x14	MONO (EGA_MONO VGA_MONO)
c80x25	80	25	8x8	CGA (EGA VGA)
em80x25	80	25	8x14	EGA_MONO (VGA_MONO)
e80x25	80	25	8x14	EGA (VGA)
vm80x25	80	25	8x16	VGA_MONO
v80x25	80	25	8x16	VGA
e80x43	80	43	8x14	EGA (VGA)

Graphics Modes		
Command	Pixel Resolution	Colors
mode5	320x200	4
mode6	640x200	2
modeD	320x200	16
modeE	640x200	16
modeF	640x350	2 (mono)
mode10	640x350	16
mode11	640x480	2
mode12	640x480	16
mode13	320x200	256

See Also

screen(HW)

Notes

The *internal* and *external* commands do not work with all types of portables.

vmstat

report paging and system statistics

Syntax

vmstat [**-fs**] [**-n** namelist] [**-l** lines] [interval [count]]

Description

vmstat reports some statistics kept by the system on processes, demand paging, and cpu and trap activity. Three types of reports are available:

(default)
A summary of the number of processes in various states, paging activity, system activity, and cpu cycle consumption.

-f Number of *fork*(S)'s done.

-s A verbose listing of paging and trap activity.

If no *interval* or *count* is specified, the totals since system bootup are displayed.

If an *interval* is given, the number of events that have occurred in the last *interval* seconds is shown. If no *count* is specified, this display is repeated forever every *interval* seconds. Otherwise, when a *count* is also specified, the information is displayed *count* times.

Other flags that may be specified include:

-n *namelist*
Use file *namelist* as an alternate symbol table instead of **/unix**.

-l *lines*
For the default display, repeat the header every *lines* reports (default is **20**).

The fields in the default report are:

procs
The number of processes which are:

r In the run queue.

b Blocked waiting for resources.

w Swapped out.

These values always reflect the current situation, even if the totals since boot are being displayed.

paging
Reports on the performance of the demand paging system. Unless the totals since boot are being displayed, this information is averaged over the proceeding *interval* seconds:

frs Free swap space.

dmd
Demand zero and demand fill pages.

sw Pages on swap.

cch
Pages in cache.

fil Pages on file.

pft
Protection faults.

frp
Pages freed.

pos
Processes swapped out successfully.

pif
Processes swapped out unsuccessfully.

rso
Regions swapped out.

rsi Regions swapped in.

system
Reports on the general system activity. Unless the totals since boot are being shown, these figures are averaged over the last *interval* seconds:

sy Number of system calls.

cs Number of context switches.

cpu
Percentage of cpu cycles spent in various operating modes:

us User.

su System.

id Idle.

This information may not be displayed on some systems.

The **-f** and **-s** reports are a series of lines of the form:

number description

which means that *number* of the items described by *description* happened (either since boot or in the last *interval* seconds, as appropriate). These reports should be self-explanatory.

Files

/unix
Default namelist.

/dev/kmem
Default source of statistics.

See Also

fork(S), ps(C), pstat(C)

Authorization

The behavior of this utility is affected by assignment of the **mem** authorization, which is usually reserved for system administrators. If you do not have this authorization, the command will not work. Refer to the "Using a Trusted System" chapter of the *User's Guide* for more details.

W

displays information about who is on the system and what they are doing

Syntax

w [**-hlqtw**] [**-n** namelist] [**-s** swapdev] [**-u** utmpfile] [users...]

Description

w prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, and the number of users logged onto the system. On systems that maintain the necessary data, the heading line also shows load averages. Load averages are the number of processes in the run queue averaged over 1, 5, and 15 minutes.

The options are:

-h Don't print the heading or title lines.

-l
Long format (default): For each user, *w* outputs the user's login name, the terminal or pseudo terminal the user is currently using, when the user logged onto the system, the number of minutes the user has been idle (how much time has expired since the user last typed anything), the CPU time used by all processes and their children attached to the terminal, the CPU time used by the currently active process, and the name and arguments of the currently active process.

-q Quick format: For each user, *w* outputs the user's login name, the terminal or pseudo terminal the user is currently using, the number of minutes the user has been idle, and the name of the currently active process.

-t
Only the heading line is output (equivalent to uptime(C)).

-w Both the heading line and the summary of users is output.

-n *namelist*
The argument is taken as the name of an alternate *namelist* (**/unix** is the default).

-s *swapdev*
Uses the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*.

-u *utmpfile*
The file *utmpfile* is used instead of */etc/utmp* as a record of who is currently logged in.

If any *users* are given, the user summary is restricted to reporting on those users.

Files

/unix
/etc/utmp
/dev/kmem
/dev/swap

See Also

date(C), finger(C), ps(C), uptime(C), who(C), whodo(C)

Notes

The "currently active process" is only an approximation and is not always correct. Pipelines can produce strange results, as can some background processes. If *w* is completely unable to guess at the currently active process, it prints "-."

Authorization

The behavior of this utility is affected by assignment of the **mem** authorization, which is usually reserved for system administrators. If you do not have this authorization, the output will be restricted to data pertaining to your activities only. Refer to the "Using a Trusted System" chapter of the *User's Guide* for more details.

wait

awaits completion of background processes

Syntax

wait

Description

Waits until all background processes started with an ampersand (&) have finished, and reports on abnormal terminations.

wait is built in to *csh* and *sh*.

Because the *wait*(S) system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

See Also

csh(C), sh(C)

Notes

Not all the processes of a pipeline with three or more stages are children of the shell, and thus cannot be waited for.

Standards Conformance

wait is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

wc

counts lines, words and characters

Syntax

wc [**-lwc**] [names]

Description

wc counts lines, words and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **-lwc**.

When *names* are specified on the command line, they are printed along with the counts.

Standards Conformance

wc is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

what

identifies files

Syntax

what files

Description

what searches the given files for all occurrences of the pattern @(#) and prints out what follows until the first tilde (˜), greater-than sign (>), new-line, backslash (\) or null character. The SCCS command *get*(CP) substitutes this string as part of the @(#) string.

For example, if the shell procedure in file **print** contains

```
#   @(#)this is the print program
#   @(#)syntax: print [files]
pr $* | lpr
```

then the command

what print

displays the name of the file **print** and the identifying strings in that file:

```
print:
        this is the print program
        syntax: print [files]
```

what is intended to be used with the *get*(CP) command, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

See Also

admin(CP), get(CP)

Standards Conformance

what is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

who

lists who is on the system

Syntax

who [**-uATHldtasqbrp**] [file]

who am i

who am I

Description

who can list the user's name, terminal line, login time, and the elapsed time since activity occurred on the line; it also lists the process ID of the command interpreter (shell) for each current user. It examines the **/tcb/files/inittab** file to obtain information for the Comments column, and **/etc/utmp** to obtain all other information. If *file* is given, that file is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

who with the **am i** or **am I** option identifies the invoking user.

Except for the default **-s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

-u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process ID of the user's shell. The *comment* is the comment field. It can contain information about where the terminal is located, the telephone number of the dataset, the type of terminal if hard-wired, etc.

-A This option displays UNIX accounting information.

-T This option is the same as the **-u** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A plus character (+) appears if the terminal is writable by anyone; a minus character (-) appears if it is not. **Root** can write to all lines having a plus character (+) or a minus character (-) in the *state* field. If a bad line is encountered, a question mark (?) is displayed.

-l This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

-H This option displays column headings above the regular output.

-q This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

-d This option displays all processes that have expired and have not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(S)), of the dead process. This can be useful in determining why a process terminated.

-t This option indicates the last change to the system clock (via the *date*(C) command) by **root**. See *su*(C).

-a This option processes the **/etc/utmp** file or the named *file* with all options turned on.

-s This option is the default and lists only the *name*, *line*, and *time* fields.

-p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/tcb/files/inittab**. The *state*, *line*, and *idle* fields have no meaning. The *comment* field shows the *id* field of the line from **/tcb/files/inittab** that spawned this process. See *inittab*(F).

-b This option indicates the time and date of the last reboot.

-r This option indicates the current *run-level* of the *init* process. In addition, it produces the process termination status, process id, and process exit status [see *utmp*(F)] under the *idle*, *pid*, and *comment* headings, respectively.

Files

/etc/utmp
/etc/wtmp
/tcb/files/inittab

See Also

date(C), login(M), mesg(C), su(C), utmp(F), inittab(F), wait(S)

Standards Conformance

who is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

whodo

determines who is doing what

Syntax

/etc/whodo

Description

whodo produces merged, reformatted, and dated output from the *who*(C) and *ps*(C) commands.

See Also

ps(C), who(C)

Authorization

The behavior of this utility is affected by assignment of the mem authorization, which is usually reserved for system administrators. If you do not have this authorization, the output will be restricted to data pertaining to your activities only. Refer to the "Using a Trusted System" chapter of the *User's Guide* for more details.

Standards Conformance

whodo is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

write

writes to another user

Syntax

write user [tty]

Description

write copies lines from your terminal to that of another user. When first called, it sends the message:

`Message from` *your-logname your-tty* `. . .`

The recipient of the message should write back at this point. Communication continues until an end-of-file is read from the terminal or an interrupt is sent. At that point, *write* displays:

```
(end of message)
```

on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *tty* argument may be used to indicate the appropriate terminal.

Permission to write may be denied or granted by use of the *mesg*(C) command. At the outset, writing is allowed. Certain commands, in particular *nroff*(CT) and *pr*(C), disallow messages in order to prevent messy output.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for him or her to write back before starting to send. Each party should end each message with a distinctive signal (**(o)** for "over" is conventional), indicating that the other may reply; **(oo)** for "over and out" is suggested when conversation is to be terminated.

Files

/etc/utmp	To find user
/bin/sh	To execute **!**

See Also

mail(C), mesg(C), who(C)

Standards Conformance

write is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

x286emul

emulate XENIX 80286

Syntax

x286emul [arg . . .] **prog286**

Description

x286emul is an emulator that allows programs from XENIX System V/286 Release 2.3 or XENIX System V/286 Release 2.3.2 on the Intel 80286 to run on the Intel 80386 processor under UNIX System V Release 3.2 or later.

Altos UNIX System V recognizes an attempt to *exec*(S) a 286 program, and automatically *exec*'s the 286 emulator with the 286 program name as an additional argument. It is not necessary to specify the *x286emul* emulator on the command line. The 286 programs can be invoked using the same command format as on the XENIX System V/286.

x286emul reads the 286 program's text and data into memory and maps them through the LDT (via *sysi86*(S)) as 286 text and data segments. It also fills in the jam area, which is used by XENIX programs to do system calls and signal returns. *x286emul* starts the 286 program by jumping to its entry point.

When the 286 program attempts to do a system call, *x286emul* takes control. It does any conversions needed between the 286 system call and the equivalent 386 system call, and performs the 386 system call. The results are converted to the form the 286 program expects, and the 286 program is resumed.

The following are some of the differences between a program running on a 286 and a 286 program using *x286emul* on a 386:

> Attempts to unlink or write on the 286 program will fail on the 286 with ETXTBSY. Under *x286emul*, they will not fail.
>
> *ptrace*(S) is not supported under *x286emul*.
>
> The 286 program must be readable for the emulator to read it.

Files

/bin/x286emul
The emulator must have this name and be in **/bin** if it is to be automatically invoked when *exec* (S) is used on a 286 program.

xargs

constructs and executes commands

Syntax

xargs [flags] [command [initial-arguments]]

Description

xargs combines the fixed *initial-arguments* with arguments read from the standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for using the shell **$PATH** variable. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or newlines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted: Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings, a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (exception: see **-i** flag). Flags **-i**, **-l**, and **-n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **-l** vs. **-n**), the last flag has precedence. *Flag* values are:

-l*number* *Command* is executed for each *number* lines of nonempty arguments from the standard input. This is instead of the default single line of input for each *command*. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first newline *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next nonempty line. If *number* is omitted, 1 is assumed. Option **-x** is forced.

-i*replstr* Insert mode: *command* is executed for each line from the standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **-x** is also forced. **{}** is assumed for *replstr* if not specified.

-n*number* Executes *command* , using as many standard input arguments as possible, up to the *number* of arguments maximum. Fewer arguments are used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **-x** is also coded, each *number* of arguments must fit in the *size* limitation, or *xargs* terminates execution.

-t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

-p Prompt mode: The user is prompted whether to execute *command* at each invocation. Trace mode (**-t**) is turned on to display the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything), will execute the command; anything else, including a carriage return, skips that particular invocation of *command*.

-x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **-x** is forced by the options **-i** and **-l**. When neither of the options **-i**, **-l**, or **-n** are coded, the total length of all arguments must be within the *size* limit.

-s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **-s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-e*eofstr* *Eofstr* is taken as the logical end-of-file string. Underscore (_) is assumed for the logical **EOF** string if **-e** is not coded. **-e** with no *eofstr* coded turns off the logical **EOF** string capability (underscore is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs terminates if it either receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(C)) with an appropriate value to avoid accidentally returning with **-1**.

Examples

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{ } $2/{ }
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is prompted to enter which files in the current directory are to be printed and prints them one at a time:

```
ls | xargs -p -l lpr
```

Or many at a time:

```
ls | xargs -p -l | xargs lpr
```

The following will execute *diff*(C) with successive pairs of arguments originally entered as shell arguments:

```
echo $* | xargs -n2 diff
```

Standards Conformance

xargs is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

xprcat

use transparent printer over modem line

Syntax

xprcat **[-bf]** [*file*] ...

Description

The *xprcat* utility "transparently" prints files over modem lines. The modem port must have an entry in the *xprtab*(F) file. Entries may be made in the *xprtab* file using *pcu*(ADM). The TERM environment variable must be set to a valid *terminfo*(M) terminal name that supports the **mc5** (printer_on) and **mc4** (printer_off) *terminfo*(M) capabilities. Only one *xprcat* request per modem port may be posted at a time.

If no input file is given, or if a single dash (-) is given, *xprcat* reads from the standard input. The process ID of the *xprcat* job is printed on the standard output. The process ID may be used to kill the transparent print request. The options are:

-b A formfeed is printed before the first file.

-f A formfeed is printed after each file.

Files

/etc/xprtab
/dev/xpr/xpr*
/dev/tty*

See Also

pcu(ADM), xprsetup(ADM), xprtab(F)

Value Added

xprcat is an extension of AT&T System V provided by Altos UNIX System V.

xtod

change file format from UNIX to MS-DOS

Syntax

xtod [filename] > [output.file]

Description

The *xtod* command converts a file from UNIX format to MS-DOS format. The MS-DOS files terminate a line of text with a carriage return and a linefeed, while UNIX files terminate a line with a linefeed only. Also MS-DOS places a ⟨CTL⟩z at the end of a file, while UNIX does not. Some programs and utilities are sensitive to this difference and some are not. If a text or data file is not being interpreted correctly, use the *dtox* and *xtod* conversion utilities. The *xtod* command adds the extra carriage return to the end of each line and adds the ⟨CTL⟩z to the end of the file. This utility is not required for converting binary object files.

If no filename is specified on the command line, *xtod* takes input from standard input. Output of the utility goes to standard output.

See Also

dtox(C)

yes

prints string repeatedly

Syntax

yes [string]

Description

yes repeatedly outputs ‘‘y’’, or if a single string argument is given, *arg* is output repeatedly. The command will continue indefinitely unless aborted. Useful in pipes to commands that prompt for input and require a ‘‘y’’ response for a yes. In this case, *yes* terminates when the command it pipes to terminates, so that no infinite loop occurs.

Altos UNIX® System V/386
Release 3.2

(M) Miscellaneous

Contents

Miscellaneous **(M)**

numtbl	create a numeric locale table
powerfail	perform power failure shutdown service
prof	profile within a function
profile	sets up an environment at login time
promain, nopromain	restrict the execution domain of a program
restart	perform power failure recovery service
rmb	remove extra blank lines from a file
streamio	STREAMS ioctl commands
subsystem	security subsystem component description
sxt	pseudo-device driver
systty	system maintenance device
term	conventional names for terminals
terminals	list of supported terminals
terminfo	terminal capability data base
termio	general terminal interface
termios	POSIX general terminal interface
timod	Transport Interface cooperating STREAMS module
timtbl	create a time locale table
tirdwr	Transport Interface read/write interface STREAMS module
trchan	translate character sets
tty	special terminal interface
tz	time zone environment variable
values	machine-dependent values
xtproto	multiplexed channels protocol used by xt(HW) driver

Intro

introduction to miscellaneous features and files

Description

This section contains miscellaneous information useful in maintaining the system. Included are descriptions of files, devices, tables and programs that are important in maintaining the entire system.

aom

Altos Office Manager Menu System

Syntax

aom

Description

The Altos Office Manager Menu System (referred to as AOM) provides a user-friendly interface to applications (such as Uniplex and Informix) and specific system utilities (such as creating directories, backing up files, and listing files).

When you enter AOM, you will see four squares containing menus for the applications and utilities that are installed on your system. You will also see three lines of information about the menus.

If you have installed an application with an AOM menu, you can link the menu into the AOM by using the add command on the Menu Manager Screen.

For more details see Altos manual *Using the AOM Menu System*, part number 690-23814-*xxx*.

Files

/usr/aom/AOMcap
/usr/aom/aomtext
/usr/aom/aom.msgs
/usr/aom/defupdate
/usr/aom/form.format
/usr/aom/form.recover
/usr/aom/aomnames
/usr/aom/aomplanes
/usr/aom/creatext
/usr/aom/form.archive
/usr/aom/form.fsck
/usr/aom/setmsg

See Also

fconvert(M), mconvert(M)

ascii

map of the ASCII character set

Description

ascii is a map of the 7-bit ASCII character set. It lists both octal and hexadecimal equivalents of each character. It contains:

Octal							
000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

Hexadecimal							
00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

The extended 8-bit ASCII character set is shown here, again with the octal and hexadecimal value of each character. The *mapchan*(C) utility allows access to these characters. Display of these characters is dependent on the capabilities of the hardware device. (A ▒ indicates an unassigned character.)

Octal							
200 ▒	201 ▒	202 ▒	203 ▒	204 ind	205 nel	206 ssa	207 esa
210 hts	211 htj	212 vts	213 pld	214 plu	215 ri	216 ss2	217 ss3
220 dcs	221 pu1	222 pu2	223 sts	224 cch	225 mw	226 spa	227 epa
230 ▒	231 ▒	232 ▒	233 csi	234 st	235 osc	236 pm	237 apc
240 nbsp	241 ¡	242 ¢	243 £	244 ¤	245 ¥	246 ¦	247 §
250 ¨	251 ©	252 ª	253 «	254 ¬	255 shy	256 ®	257 ¯
260 °	261 ±	262 ²	263 ³	264 ´	265 µ	266 ¶	267 ·
270 ¸	271 ¹	272 º	273 »	274 ¼	275 ½	276 ¾	277 ¿
300 À	301 Á	302 Â	303 Ã	304 Ä	305 Å	306 Æ	307 Ç
310 È	311 É	312 Ê	313 Ë	314 Ì	315 Í	316 Î	317 Ï
320 Ð	321 Ñ	322 Ò	323 Ó	324 Ô	325 Õ	326 Ö	327 ▒
330 Ø	331 Ù	332 Ú	333 Û	334 Ü	335 Ý	336 Þ	337 ß
340 à	341 á	342 â	343 ã	344 å	345 æ	346 æ	347 ç
350 è	351 é	352 ê	353 ë	354 ì	355 í	356 î	357 ï
360 đ	361 ñ	362 ò	363 ó	364 ô	365 õ	366 ö	367 ▒
370 ø	371 ù	372 ú	373 û	374 ü	375 ý	376 þ	377 ÿ

Hexadecimal							
80 ▒	81 ▒	82 ▒	83 ▒	84 ind	85 nel	86 ssa	87 esa
88 hts	89 htj	8a vts	8b pld	8c plu	8d ri	8e ss2	8f ss3
90 dcs	91 pu1	92 pu2	93 sts	94 cch	95 mw	96 spa	97 epa
98 ▒	99 ▒	9a ▒	9b csi	9c st	9d osc	9e pm	9f apc
a0 nbsp	a1 ¡	a2 ¢	a3 £	a4 ¤	a5 ¥	a6 ¦	a7 §
a8 ¨	a9 ©	aa ª	ab «	ac ¬	ad shy	ae ®	af ¯
b0 °	b1 ±	b2 ²	b3 ³	b4 ´	b5 µ	b6 ¶	b7 ·
b8 ¸	b9 ¹	ba º	bb »	bc ¼	bd ½	be ¾	bf ¿
c0 À	c1 Á	c2 Â	c3 Ã	c4 Ä	c5 Å	c6 Æ	c7 Ç
c8 È	c9 É	ca ê	cb Ë	cc Ì	cd Í	ce Î	cf Ï
d0 Ð	d1 Ñ	d2 Ò	d3 Ó	d4 Ô	d5 Õ	d6 Ö	d7 ▒
d8 Ø	d9 Ù	da Ú	db Û	dc Ü	dd Ý	de Þ	df ß
e0 à	e1 á	e2 â	e3 ã	e4 ä	e5 å	e6 æ	e7 ç
e8 è	e9 é	ea ê	eb ë	ec ì	ed í	ee î	ef ï
f0 đ	f1 ñ	f2 ò	f3 ó	f4 ô	f5 õ	f6 ö	f7 ▒
f8 ø	f9 ù	fa ú	fb û	fc ü	fd ý	fe þ	ff ÿ

Files

/usr/pub/ascii

chrtbl

create a ctype locale table

Syntax

chrtbl [*specfile*]

Description

The utility *chrtbl* is provided to allow new LC_CTYPE locales to be defined; It reads a specification file, containing definitions of the attributes of characters in a particular character set, and produces a binary table file, to be read by *setlocale*(S), which determines the behavior of the *ctype*(S) and *conv*(S) routines.

The information supplied in the specification file consists of lines in the following format:

char type conv

The three fields, which are separated by space or tab characters, have the following meanings and syntax:

char This is the character which is being defined. It may be specified in one of six different ways (the following examples all specify the ASCII character "A"):

65	decimal
0101	octal
0x41	hexadecimal
'A'	quoted character
'\101'	quoted octal
'\x41'	quoted hexadecimal

type This specifies the classification of the character, as reported by the *ctype*(S) routines. There are 7 basic classifications:

C	iscntrl
D	sdigit
L	islower
P	ispunct
S	isspace

U	isupper
X	isxdigit

Other ctype macros use combinations of these 7 basic classifications. Zero, one or more of these classification letters can be specified, in any order, although only certain combinations are logically reasonable, as follows:

C	control character
CS	spacing control character
U	uppercase alphabetic
UX	uppercase alphabetic hex digit
UL	dual case character
L	lowercase alphabetic
LX	lowercase alphabetic hex digit
DX	decimal and hex digit
S	spacing character
P	punctuation (all other printing chars)
blank	undefined (all classifications false)

conv This optional field specifies the corresponding upper case character for a lower case character, or the corresponding lower case character for an upper case character. Dual case characters should have their own values repeated in this field.

The syntax is as for the *char* field.

All characters following a hash (#) are treated as a comment and ignored up to the end of the line, unless the hash is within a quoted character.

The initial LC_CTYPE table used is that for the *ascii*(M) character set, with the entries for the higher 128 characters (0x80 - 0xff) set to zero (i.e. all classifications false). Thus an empty specification file will result in a table for US ASCII . Any specifications found in the input to *chrtbl* will overwrite the specifications for that character only, thus additions and modifications to the ASCII table can be made without respecifying those characters which are unchanged.

The binary table output is placed in a file named *ctype*, within the current directory. This file should be copied or linked to the correct place in the *setlocale* file tree (see *locale*(M)). To prevent accidental corruption of the output data, the file is created with no write permission; if the *chrtbl* utility is run in a directory containing a write-protected "ctype" file, the utility will ask if the existing file should be replaced; any response other than "yes" or "y" will cause *chrtbl* to terminate without overwriting the existing file.

If the *specfile* argument is missing, the specification information is read from the standard input.

Diagnostics

If the input table file cannot be opened for reading, processing will terminate with the error message, "Cannot open specification file".

Any lines in the specification file which are syntactically incorrect will cause an error message to be issued to the standard error output, specifying the line number on which the error was detected. The line will be ignored, and processing will continue.

If the output file, "ctype", cannot be opened for writing, processing will terminate with the error message, "Cannot create table file."

Any error conditions encountered will cause the program to exit with a non-zero return code; successful completion is indicated with a zero return code.

Specification File Format

The *chrtbl* specification file has the following format (the order of the specifications is not significant):

```
#
# chrtbl file for TVI 7-bit Spanish character set
# Note that only non-ASCII characters need be specified
#
'@'     P           # inverted ?
'['     L   ']'     # n tilde
'\\'    P           # inverted !
']'     U   '['     # N tilde
'~'     P           # degree sign
```

Files

/usr/include/ctype.h

See Also

ascii(M), conv(S), ctype(S), locale(M), setlocale(S)

Value Added

chrtbl is an extension of AT&T System V provided in Altos UNIX System V.

clone

open any minor device on a STREAMS driver

Description

clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to *clone* during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate *stream* to a previously unused minor device.

The *clone* driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls [including *close*(S)] require no further involvement of *clone*.

clone will generate an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

Warnings

Multiple opens of the same minor device cannot be done through the *clone* interface. Executing *stat*(S) on the file system node for a cloned device yields a different result from executing *fstat*(S) using a file descriptor obtained from opening the node.

See Also

log(M)

STREAMS Programmer's Guide

coltbl

create a collation locale table

Syntax

coltbl [*specfile*]

Description

The utility *coltbl* is provided to allow LC_COLLATE locales to be defined. It reads in a specification file (or standard input if *specfile* is not defined), containing defintions for a particular locale's collation ordering, and produces a concise format table file, to be read by *setlocale*(S).

In general, characters may be specified in one of six different ways (the following examples all specify the ASCII character "A"):

65	decimal
0101	octal
0x41	hexadecimal
'A'	quoted character
'\101'	quoted octal
'\x41'	quoted hexadecimal

The information in the specification file is to an extent free format. A particular type of definition is started by one of the following key-words:

PRIM:
ZERO:
EQUIV:
DOUBLE:

The keywords, *PRIM:*, *ZERO:* and *EQUIV:*, are concerned directly with the setting of the collation ordering of characters

A group of characters which are to be collated as equal, unless all other characters in a pair of strings are also equal, are grouped together with the *PRIM:* keyword. The position of a particular group in the specification file is significant as far as the collation ordering is concerned. Collating elements following the *PRIM:* keyword are separated by white spaces. A two character collating element can be specified here by *(a b)*, where *a* and *b* are the two characters making up the sequence. The order of the collating elements defined in one group is significant in secondary collation ordering. It is also possible to define a range of characters, for example:

PRIM: 'a' - 'z'

Collating elements following the *ZERO:* keyword, are to be ignored when collating. The format of the definitions is the same as with *PRIM:* . Ranges of characters can also be defined, as for example:

ZERO: 0x80 - 0x9f

EQUIV: is used to give two collating elements identical positions in the collation ordering. The syntax is:

EQUIV: a = b

where *a* and *b* are the two equal collating elements. There can be only one definition for each occurrence of this keyword.

Single characters which are to be collated as two characters, for example the German sharp s, are defined with the *DOUBLE:* keyword. The syntax is:

DOUBLE: a = (b c)

where *a* is the single character, and *b* and *c* are the two characters in the collating sequence. There can be only one definition for each occurrence of this keyword. The single character *a* must not also appear after a *PRIM:* , a *ZERO:* or a *EQUIV:* keyword.

All characters following the hash character are treated as a comment and ignored up to the end of the line, unless the hash is within a quoted string.

The concise format locale table is placed in a file named *collate* in the current directory. This file should be copied or moved to the correct place in the *setlocale* (S) file tree (see *locale* (M)). To prevent accidental corruption of the output data, the file is created with no write permission; if the *coltbl* utility is run in a directory containing a write-protected *collate* file, the utility will ask if the existing file should be replaced - any response other than "yes" or "y" will cause *coltbl* to terminate without overwriting the existing file.

See Also

chrtbl(M), collation(S), locale(M), numtbl(M), mestbl(M), montbl(M), timtbl(M), setlocale(S)

Diagnostics

All error messages printed are self explanatory.

Value Added

coltbl is an extension of AT&T System V provided in Altos UNIX System V.

console

system console device

Description

The file **/dev/console** is the device used by the system administrator for system maintenance (single-user) operations. It is the **tty** to which the first default shell is attached.

The system *console* device can be either a terminal (a serial adapter device, **tty1a**) or a sytem keyboard display adapter monitor (**tty01**).

Many programs, such as the Altos UNIX System V kernel, redirect error messages to **/dev/console**. Initially **/dev/console** is linked to **/dev/systty**.

Files

/dev/console

See Also

boot(HW), screen (HW), systty(M), tty(M)

Notes

/dev/console should not be enabled, instead either the the display adapter (**tty01**) or the serial adapter device (**tty1a**) should be enabled.

A serial console cannot be attached to a multiport card or one that uses special drivers; it must be on a standard COM1 card.

Standards Conformance

console is conformant with:

The X/Open Portability Guide II of January 1987.

cvtcoff

convert 386 COFF files to XENIX format

Syntax

cvtcoff [**-v**] [**-o** outfile] coff-file

Description

cvtcoff converts 386 Common Object Format Files (COFF) to the appropriate XENIX file format. If the file specified is a relocatable object module it is converted to Microsoft OMF format. If it is an executable binary it is converted to x.out format.

If the file is a UNIX System V archive, it is converted to XENIX archive format and each file in the archive is converted as appropriate. Any files in the archive which are not in 386 COFF format are copied to the new archive unchanged. *cvtcoff* also creates a XENIX format __.SYMDEF symbol directory for the new archive.

Options are:

-v Verbose mode. The name of each member of an archive is displayed as it is converted.

-o Output file name. The output file will be named **a.out** by default if no output file name is given.

Notes

Only essential symbol table information is converted. Source line numbers and additional symbol information for use by the symbolic debugger sdb will be ignored.

Note that *cvtcoff* only converts 386 COFF files. It is not possible to convert 286 COFF files.

Files

x.out Default output file

See Also

cvtomf(M), 86rel(F), a.out(F), ar(F)

Value Added

cvtcoff is an extension of AT&T System V provided in Altos UNIX System V.

cvtomf

convert XENIX files to UNIX COFF format

Syntax

cvtomf omf-file

Description

cvtomf converts XENIX file format to 386 Common Object Format Files (COFF). If the file specified is a relocatable object module it is converted to COFF format.

Notes

Note that *cvtomf* only converts 386 XENIX files. It is not possible to convert 286 XENIX files.

Be sure and consult the *XENIX Compatibility Guide* for possible pitfalls relating to file conversion.

See Also

cvtcoff(M), 86rel(F), a.out(F), ar(F), cc(CP)

Value Added

cvtomf is an extension of AT&T System V provided in Altos UNIX System V.

daemon.mn

micnet mailer daemon

Syntax

/usr/lib/mail/daemon.mn [-ex]

Description

The mailer daemon performs the "backend" networking functions of the *mail*, *rcp*, and *remote* commands by establishing and servicing the serial communication link between computers in a Micnet network.

When invoked, the daemon creates multiple copies of itself, one copy for each serial line used in the network. Each copy opens the serial line, creates a startup message for the LOG file, and waits for a response from the daemon at the other end. The startup message lists the names of the machines to be connected, the serial line to be used, and the current date and time. If the daemon receives a correct response, it establishes the serial link and adds the message "first handshake complete" to the LOG file. If there is no response, the daemon waits indefinitely.

If invoked with the -x switch, the daemon records each transmission in the LOG file. A transmission entry shows the direction of the transmission (tx for transmit, rx for receive), the number of bytes transmitted, the elapsed time for the transmission (in minutes and seconds), and the time of day of the transmission (in hours, minutes, and seconds). Each entry has the form:

direction byte_count elapsed_time time_of_day

The daemon also records the date and time every hour. The date and time have the same format as described for the *date* command.

If invoked with the -e switch, the daemon records all transmission errors in the LOG file. An error entry shows the cause of the error preceded by the name of the daemon subroutine which detected the error.

The mailer daemon is normally invoked by the *start* option of the *netutil* command and is stopped by the *stop* option.

During the normal course of execution, the mailer daemon uses several files in the **/usr/spool/micnet/remote** directory. These files provide storage for LOG entries, commands issued by the *remote*(C) command, and a list of processes under daemon control.

Files

/usr/lib/mail/daemon.mn

/usr/spool/micnet/remote/*/LOG

/usr/spool/micnet/remote/*/mn

/usr/spool/micnet/remote/local/mn*

/usr/spool/micnet/remote/lock

/usr/spool/micnet/remote/pids

See Also

netutil(ADM)

environ

the user environment

Description

The user environment is a collection of information about a user, such as login directory, mailbox, and terminal type. The environment is stored in special "environment variables," which can be assigned character values, such as names of files, directories, and terminals. These variables are automatically made available to programs and commands invoked by the user. The commands can then use the values to access the user's files and terminal.

The following is a short list of commonly used environment variables.

PATH — Defines the search path for the directories containing commands. The system searches these directories whenever a user types a command without giving a full pathname. The search path is one or more directory names separated by colons (:). Initially, PATH is set to :/bin:/usr/bin.

HOME — Names the user's login directory. Initially, HOME is set to the login directory given in the user's **passwd** file entry.

EXINIT — Used to set *vi* options and define *vi* abbreviations and mappings. For Bourne Shell users, the syntax is:

EXINIT = 'set ***options*'**

For C-Shell users, the syntax is:

setenv EXINIT 'set ***options*'**

For example, a C-Shell user might place the following command in **$HOME/.cshrc**:

setenv EXINIT 'set wm=24 | map g 1G'

This would automatically set *vi's* wrapmargin option to 24 and would define the "g" key to move to the top of the file (just as "G" moves to the bottom of the file).

You can set more than one option with the same **set** command. If you define abbreviations or mappings with this environment variable, you must separate the **abbr** and **map** commands from the **set** command and from each other with a bar (|). The function of the bar is similar to that of the semicolon that separates commands on a shell command line.

If you are defining many customizations, you might prefer to use the **.exrc** file, where each command can be listed one per line (see *vi*(C)).

TERM Defines the type of terminal being used. This information is used by commands such as *more*(C) which rely on information about the capabilities of the user's terminal. The variable may be set to any valid terminal name (see *terminals*(M)) directly or by using the *tset*(C) command.

TZ Defines time zone information. This information is used by *date*(C) to display the appropriate time. The variable may have any value of the form:

xxx*nzzzs;* **start/***time*, **end/***time*

where **xxx** is standard local time zone abbreviation (1-9 characters), *n* is the standard time zone difference from GMT, and may be given as hh:mm:ss (hours:minutes:seconds), **zzz** is the summertime local time zone abbreviation of 1-9 characters (if any), *s* is the summertime time zone difference from GMT, and may be given as hh:mm:ss (hours:minutes:seconds), **start** and **end** specify the day to begin and end summertime based on one of four rules, and **time** is the time of day the change to or from summertime occurs. The rules for specifying **start** and **end** are:

J*n*	1 based Julian day *n*
n	0 based Julian day *n*
W*n.d*	*n*th day of week *d*
M*m.n.d*	*n*th day of week *d* in month *m*

For example:

```
EST5:00:00EDT4:00:00;M4.1.0/2:00:00,M10.5.0/2:00:00.
```

Refer to the *tz*(M) manual page for more on *TZ*.

HZ Defines, with a numerical value, the number of clock interrupts per second. The value of this variable is dependent on the hardware, and configured in the file **etc/default/login**. If *HZ* is not defined, programs which depend on this hertz value, such as *prof*(CP) and *times*(S), will not run.

LANG Represents the international *locale* in the format *language_territory.codeset*. This is used by *setlocale* (S) to establish the default *locale* on program startup.

Individual locale-specific functions can be affected independently using the following optional environment variables:

LC_CTYPE Locale affecting character classification routines (*ctype(S)*).

LC_NUMERIC
Locale affecting numeric formatting.

LC_TIME Locale affecting time and date format.

LC_COLLATE
Locale affecting collation/sorting sequence.

LC_MESSAGES
Locale affecting message language.

LC_MONETARY
Locale affecting currency formatting.

The environment can be changed by assigning a new value to a variable. An assignment has the form:

name=value

For example, the assignment:

TERM=h29

sets the TERM variable to the value "h29". The new value can be "exported" to each subsequent invocation of a shell by exporting the variable with the *export* command (see *sh*(C)) or by using the *env*(C) command.

You may also add variables to the environment, but you must be sure that the new names do not conflict with exported shell variables such as MAIL, PS1, PS2, and IFS. Placing assignments in the **.profile** file is a useful way to change the environment automatically before a session begins.

Note that the environment is made available to all programs as an array of strings. Each string has the form:

name=value

where the *name* is the name of an exported variable and the *value* is the variable's current value. For programs started with a *exec*(S) call, the environment is available through the external pointer *environ*. For other programs, individual variables in environment are available through *getenv*(S) calls.

See Also

env(C), exec(S), getenv(S) setlocale(S), locale(M), login(M), profile(M), sh(C)

Standards Conformance

environ is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

error

kernel error output device

Description

System error messages are collected and made available to error logging daemons through the **/dev/error** device. **/dev/error** is a read-only device which returns one error per read and no EOF character. The **/etc/rc2** scripts use a utility to read messages from **/dev/error** and write them to the system error log file **/usr/adm/messages**:

```
/etc/logger /dev/error /usr/adm/messages &
```

Any process can read **/dev/error** or arrange to be signaled when errors are queued in **/dev/error**. The following *ioctl* causes the error device to signal the process with **SIGUSR1** when an error message is queued in **/dev/error**.

```
#include <signal.h>
#include <sys/eio.h>
#include <fcntl.h>
...
int fd;
...
fd = open("/dev/error", O_RDONLY);
ioctl(fd, EMSG_SIG, SIGUSR1);
```

Before exiting, the process must return **/dev/error** to its normal state. Do this with the following *ioctl*:

```
...
ioctl(fd,EMSG_NOSIG, 0);
...
```

Panic error messages are not logged in **/dev/error**.

Files

/dev/error

See Also

messages(M)

fcntl

file control options

Syntax

#include <fcntl.h>

Description

The *fcntl*(S) function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open*(S).

```
/* Flag values accessible to open(S) and fcntl(S) */
/*  (The first three can only be set by open) */
#define O_RDONLY 0
#define O_WRONLY 1
#define O_RDWR   2
#define O_NDELAY 04     /* Non-blocking I/O */
#define O_APPEND 010    /* append (writes guaranteed at the end) */
#define O_SYNC   020    /* synchronous write option */

/* Flag values accessible only to open(S) */
#define O_CREAT  00400  /* open with file create (uses third open arg)*/
#define O_TRUNC  01000  /* open with truncation */
#define O_EXCL   02000  /* exclusive open */

/* fcntl(S) requests */
#define F_DUPFD  0      /* Duplicate fildes */
#define F_GETFD  1      /* Get fildes flags */
#define F_SETFD  2      /* Set fildes flags */
#define F_GETFL  3      /* Get file flags */
#define F_SETFL  4      /* Set file flags */
#define F_GETLK  5      /* Get file lock */
#define F_SETLK  6      /* Set file lock */
#define F_SETLKW 7      /* Set file lock and wait */
#define F_CHKFL  8      /* Check legality of file flag changes */

/* file segment locking control structure */
struct flock {
      short l_type;
      short l_whence;
      long  l_start;
      long  l_len;      /* if 0 then until EOF */
      short l_sysid;    /* returned with F_GETLK*/
      short l_pid;      /* returned with F_GETLK*/
}
```

```
/* file segment locking types */
#define F_RDLCK 01    /* Read lock */
#define F_WRLCK 02    /* Write lock */
#define F_UNLCK 03    /* Remove locks */
```

See Also

fcntl(S), open(S)

Standards Conformance

fcntl is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

fconvert

create form file

Syntax

fconvert input_file output_file

Description

If you used a form command in your menu file, you must create a form file.

The *fconvert* command is used for compiling a form file into an AOM form file. The input_file is the path and file name of a form file, and the output_file is the path and file name of the compiled AOM form file that can be integrated to AOM Menu System.

For more details see Altos manual *AOM Tool Kit User's Guide*, part number 690-17464-*xxx*.

Files

/usr/aom/AOMcap
/usr/aom/aomtext
/usr/aom/aom.msgs
/usr/aom/defupdate
/usr/aom/form.format
/usr/aom/form.recover
/usr/aom/aomnames
/usr/aom/aomplanes
/usr/aom/creatext
/usr/aom/form.archive
/usr/aom/form.fsck
/usr/aom/setmsg

See Also

aom(M), mconvert(M)

getclk

gets string from real-time clock

Syntax

/etc/getclk

Description

getclk get a string suitable for *date*(C) from the real-time clock and write it to stdout. Exit code 1 if it doesn't work, 0 if successful.

See Also

date(C)

getty

sets terminal type, modes, speed, and line discipline

Syntax

/etc/getty [**-h**] **-g**] [**-b** blanktime] [**-t** timeout] line [speed [type [linedisc]]]

/etc/getty -c file

Description

getty is a program that is invoked by *init*(M). It is the second process in the series, (*init-getty-login-shell*), that ultimately connects a user with the UNIX system. Initially *getty* displays the login message field for the entry it is using from **/etc/gettydefs**. *getty* reads the user's login name and invokes the *login*(M) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. The **-t** flag, plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one enters anything in the specified number of seconds. The optional second argument, *speed*, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by entering a BREAK character). The default *speed* is 9600 baud.

Type, the optional third argument, is a character string describing to *getty* what type of terminal is connected to the line in question. *getty* recognizes the following types:

none	default
ds40-1	DATASPEED terminal 40/1
tektronix,tek	TEKTRONIX
vt61	Digital Equipment vt61
vt100	Digital Equipment vt100
hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. For terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISC0**.

If the **-g** option is invoked, *getty* will use the GIF viewer to display the file **/usr/lib/gif/default.gif** on the specified terminal. The GIF viewer is a program that displays GIF format picture files. The program always displays the **default.gif** file. Common practice is to keep a library of GIF pictures in **/usr/lib/gif** and to simply link the desired GIF file to **default.gif** to have it displayed by the viewer on lines set with *getty -g*. This option is intended for use on an a virtual terminal with bit-mapped graphics capability.

The **-b** option, too, is useful only only an a virtual terminal. When specified with **-b**, *getty* will cause the screen to "blank" if nothing is typed for *blanktime* seconds. Pressing any key will re-activate a blanked screen.

When given no optional arguments, *getty* sets the *speed* of the interface to 9600 baud, specifies that raw mode will be used (awaken on every character), that echo will be suppressed, either parity allowed, that new-line characters will be converted to carriage return-line feed, and that tab expansion is performed on the standard output. It displays the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the BREAK key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(S)).

The user's name is scanned to see if it contains any lower-case alphabetic characters. *getty* suggests that the user use all lower-case characters. If the user uses upper case characters, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, *login* is **exec**'d with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment [see *login*(M)].

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it displays the values of the various flags. See *ioctl*(S) to interpret the values. Note that some values are added to the flags automatically.

Files

/etc/gettydefs
/usr/lib/gif/default.gif

See Also

init(M), login(M), uugetty(M), tty(HW), ioctl(S), gettydefs(F), inittab(F)

Notes

While *getty* understands simple single character quoting conventions, it is not possible to quote certain special control characters used by *getty*. Thus, you cannot log in via *getty* and type a **#**, **@**, **/**, **!**, **_**, back-space, **^U**, **^D**, or **&** as part of your login name or arguments. *getty* uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.

When connecting two computers using a direct connection, never invoke *getty*(M) on the ports of both machines. Instead, use *uugetty*(M).

init, telinit

process control initialization

Syntax

/etc/init [**0123456SsQqabc**]

/bin/telinit [**0123456SsQqabc**]

Description

init is a general process spawner. Its primary role is to create processes from information stored in the file **/etc/inittab** [see *inittab*(F)].

At any given time, the system is in one of eight possible run levels. A run level is a software configuration of the system under which only a selected group of processes exist. The processes spawned by *init* for each of these run levels is defined in **/etc/inittab**. *init* can be in one of eight run levels, **0-6** and S or s (run levels S and s are identical). The run level changes when a privileged user runs **/etc/init**. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was booted, telling it which run level to change to.

If the file **/etc/default/boot** contains the string **MAPKEY=YES**, *init* invokes the *mapkey* program (see *mapkey*(M)) to map the console keyboard. If the call to *mapkey* succeeds, the console is set to 8-bits no parity. If the call fails, and the string **SERIAL8=YES** appears in **/etc/default/boot**, a serial console device is assumed and set to 8-bits no parity.

The following are the arguments to *init*:

0 shut the machine down so it is safe to remove the power. Have the machine remove power if it can. This state can be executed only from the console.

1 put the system in single-user mode. Unmount all file systems except root. All user processes are killed except those connected to the console. This state can be executed only from the console.

2 put the system in multiuser mode. All multiuser environment terminal processes and daemons are spawned. This state is commonly referred to as the multiuser state.

3 Start the remote file sharing processes and daemons. Mount and advertise remote resources. Run level **3** extends multiuser mode and is known as the remote-file-sharing state.

4 This is available to be defined as an alternative multiuser environment configuration. It is not necessary for system operation and is usually not used.

5 Stop the UNIX system and go to the firmware monitor.

6 Stop the UNIX system and reboot to the state defined by the **initdefault** entry in **/etc/inittab**.

a,b,c Process only those **/etc/inittab** entries having the **a**, **b** or **c** run level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run level to change.

Q,q Re-examine **/etc/inittab**.

S,s Enter single-user mode. When this occurs, the terminal which executed this command becomes the system console (see Notes for more information about console device assignment). This is the only run level that doesn't require the existence of a properly formatted **/etc/inittab** file. If this file does not exist, then by default the only legal run level that *init* can enter is the single-user mode. When the system enters S or s, all mounted file systems remain mounted and only processes spawned by *init* are killed.

When a UNIX system is booted, *init* is invoked and the following occurs. *init* first looks in **/etc/default/boot** to determine if **autoboot** on panic is desired. *init* looks to see if DEFAULT_LEVEL=N is specified in **/etc/default/boot**. If it is, then N is the default level, otherwise, the user is prompted to see if they wish to go to multiuser or system maintenance mode (single-user mode). In the single-user state, the virtual console terminal is assigned to the user's terminal and is opened for reading and writing. The *sulogin* command, which requires the user to enter the root password, is invoked and a message is generated on the physical console saying where the virtual console has been relocated. Use either *init* or *telinit* to signal *init* to change the run level of the system. Note that if the shell is terminated (via an end-of-file), *init* will only re-initialize to the single-user state if the **/etc/inittab** file does not exist.

If a **0** through **6** is entered, *init* enters the corresponding run level. Note that, on your computer, the run levels **0**, **1**, **5**, and **6** are reserved states for shutting the system down; the run levels **2**, **3**, and **4** are available as normal operating states.

If this is the first time since power up that *init* has entered a run level other than single-user state, *init* first scans **/etc/inittab** for **boot** and **bootwait** entries [see *inittab*(F)]. These entries are performed before any other processing of **/etc/inittab** takes place, providing that the run level entered matches that of the entry. In this way, any special initialization of the operating system, such as mounting filesystems, can take place before users are allowed onto the system. *init* then scans **/etc/inittab** and executes all other entries that are to be processed for that run level.

In a multiuser environment, **/etc/inittab** is set up so that *init* will create a *getty* process for each terminal that the administrator sets up to respawn.

To spawn each process in **/etc/inittab**, *init* reads each entry and for each entry that should be respawned, it forks a child process. *init* spawns each process by forking a shell to run the job in. To set up the environment for this shell, *init* uses the **/etc/initscript** file which contains the definitions of some global variables, for example, TZ, HZ, and PATH. After it has spawned all of the processes specified by **/etc/inittab**, *init* waits for one of its descendant processes to die, a powerfail signal, or a signal from another *init* or *telinit* process to change the system's run level. When one of these conditions occurs, *init* re-examines **/etc/inittab**. New entries can be added to **/etc/inittab** at any time; however, *init* still waits for one of the above three conditions to occur before re-examining **/etc/inittab**. To get around this, an **init Q** or **init q** command wakes *init* to re-examine **/etc/inittab** immediately.

When *init* comes up at boot time and whenever the system changes from the single-user state to another run state, *init* sets the *ioctl*(S) states of the virtual console to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever the single-user state is entered.

When a run level change request is made, *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target run level. *init* waits 5 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

The shell running on each terminal will terminate when the user types an end-of-file or hangs up. When *init* receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists [see *who*(C)]. A history of the processes spawned is kept in **/etc/wtmp**.

If *init* receives a *powerfail* signal (**SIGPWR**) it scans **/etc/inittab** for special entries of the type *powerfail* and *powerwait*. These entries are invoked (if the run levels permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions during the powerdown of the operating system. Note that in the single-user states, **S** and **s**, only *powerfail* and *powerwait* entries are executed. If *init* receives a *powerfail* signal (**SIGPWR**) on a warm restart of the system after a power failure, it scans **/etc/inittab** for special entries of the type **restart**. These entries are invoked, causing some processes (e.g., network daemons) to start up.

telinit, which is linked to **/etc/init**, is used to direct the actions of *init*. It takes a one-character argument and signals *init* to take the appropriate action.

Files

/etc/default/boot
/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/etc/initscript
/dev/console
/dev/contty

See Also

disable(C), enable(C), login(M), sh(C), stty(C), who(C), getty(M), powerfail(M), restart(M), shutdown(M), sulogin(ADM), termio(HW), kill(S), gettydefs(F), inittab(F), utmp(F)

Diagnostics

If *init* finds that it is respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the **inittab** file or a program is removed that is referenced in **/etc/inittab**.

When attempting to boot the system, failure of *init* to prompt for a new run level may be because the virtual system console is linked to a device other than the physical system console.

Notes

init and *telinit* can be run only by someone who is super-user.

The S or s state must not be used indiscriminately in the **/etc/inittab** file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the **initdefault**.

The assignment of the console device may seem confusing at first. Whenever the system is rebooted, the first boot up messages will be displayed on the "normal" system console (tty01), then the prompt for going multiuser will be displayed on the the tty from which **init S** was last invoked, which could be any tty on the system. The system console device (**/dev/syscon**) remains linked to the tty from which the last **init S** is invoked. Rebooting the system does NOT reset this to tty01.

If the **/etc/initscript** file is not present, *init* will print a warning on the console and spawn the job without setting up the global environment.

The change to **/etc/gettydefs** described in the **Notes** section of the *gettydefs*(F) manual page will permit terminals to pass 8 bits to the system as long as the system is in multiuser state (run level greater than 1). When the system changes to single-user state, the *getty* is killed and the terminal attributes are lost. To permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type:

stty -istrip cs8

The **/etc/TIMEZONE** file must exist. **/etc/initscript** executes this file to set the correct TZ variable for the system.

Standards Conformance

init is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

isverify

verifies ISAM database records

Syntax

isverify [-Iilpyn] tablelist

Description

isverify detects and, if specified, repairs inconsistencies between ISAM data (**.dat**) files and index (**.idx**) files. The *isverify* utility checks that every valid record in the data file is properly represented in the index file; it also checks that every index entry points to a valid data record.

tablelist is the list of tables to be checked by *isverify*. The **.dat** and **.idx** suffixes should not be included in the *tablelist*.

Options

You can specify any of the following flags when invoking *isverify*:

-I after a system restore, an ISAM application can fail with the message:

```
Error: Incorrect SCO Runtime System installed
```

You can correct this situation by logging in as *root* and invoking **isverify -I**.

-i Check only the index file (as opposed to checking both the index and the data files) for consistency. Use this option as a quick check if you think the data files are probably not corrupted.

-l prints a long listing of the information for each defined key (index), along with the associated data record pointer. The key value for each data record is displayed by key part, along with the byte position of the data record in the data file. This information is useful only if you understand the Indexed Sequential Access Method (ISAM).

-p pauses after displaying information about each index. If you select this option, you must press the Break key before the *isverify* process continues.

-y causes *isverify* to assume a "yes" answer to each error state and to attempt to make the specified correction. It is recommended that you use this flag so that the *isverify* utility attempts to correct any discrepancies automatically.

-n causes *isverify* to assume a "no" answer to each error state and to leave the files unchanged. It also allows you see where errors are by displaying them on the screen.

Whether or not you use *isverify* with the **-l** or **-p** flags, if an error is detected, you have the option of making a correction or leaving the files unchanged. If no errors are detected, no response is required. If you choose to make a correction, *isverify* attempts to repair the files. Unless the **-y** or **-n** flags are specified on the command line, you must choose interactively whether or not to make each correction.

jagent

host control of windowing terminal

Syntax

#include <sys/jioctl.h>

ioctl (cntlfd, JAGENT, &arg)

int cntlfd
struct bagent arg

Description

The *ioctl*(S) system call, when performed on an *xt*(HW) device with the **JAGENT** request, allows a host program to send information to a windowing terminal.

ioctl has three arguments:

cntlfd — the *xt*(HW) control channel file descriptor

JAGENT — the *xt*(HW) *ioctl*(S) request to invoke a windowing terminal agent routine.

arg — the address of a *bagent* structure, defined in **<sys/jioctl.h>** as follows:

```
struct bagent {
  long  size;   /* size of src in & dest out */
  char  *src;   /* the source byte string */
  char  *dest;  /* the destination byte string */
};
```

The *src* pointer must be initialized to point to a byte string which is sent to the windowing terminal. See *layers*(M) for a list of **JAGENT** strings recognized by windowing terminals. Likewise, the *dest* pointer must be initialized to the address of a buffer to receive a byte string returned by the terminal. When *ioctl*(S) is called, the *size* argument must be set to the length of the *src* string. Upon return, *size* is set by *ioctl*(S) to the length of the destination byte string, *dest*.

See Also

ioctl(S), libwindows(S), layers(M), xt(HW)

Diagnostics

Upon successful completion, the size of the destination byte string is returned. If an error occurs, -1 is returned.

layers

protocol used between host and windowing terminal under layers(C)

Syntax

#include <sys/jioctl.h>

Description

layers are asynchronous windows supported by the operating system in a windowing terminal. Communication between the Altos UNIX System V processes and terminal processes under *layers*(C) occurs via multiplexed channels managed by the respective operating systems using a protocol as specified in *xtproto*(M).

The contents of packets transferring data between an Altos UNIX System V process and a layer are asymmetric. Data sent from the Operating System to a particular terminal process is undifferentiated and it is up to the terminal process to interpret the contents of packets.

Control information for terminal processes is sent via channel 0. Process 0 in the windowing terminal performs the designated functions on behalf of the process connected to the designated channel. These packets take the form:

command, channel

except for *timeout* and *jagent* information which take the form

command, data...

The commands are the bottom eight bits extracted from the following **ioctl**(S) codes:

JBOOT Prepare to load a new terminal program into the designated layer.

JTERM Kill the downloaded layer program and restore the default window program.

JTIMO Set the timeout parameters for the protocol. The data consist of two bytes: the value of the receive timeout in seconds and the value of the transmit timeout in seconds.

JTIMOM Set the timeout parameters for the protocol. The data consist of four bytes in two groups: the value of the receive timeout in milliseconds (the low eight bits followed by the high eight bits) and the value of the transmit timeout (in the same format).

JZOMBOOT
Like JBOOT, but do not execute the program after loading.

JAGENT Send a source byte string to the terminal agent routine and wait for a reply byte string to be returned.

The data are from a *bagent* structure [see *jagent*(M)] and consist of a one-byte size field followed by a two-byte agent command code and parameters. Two-byte integers transmitted as part of an agent command are sent with the high-order byte first. The response from the terminal is generally identical to the command packet, with the two command bytes replaced by the return code: **0** for success, **-1** for failure. Note that the routines in the *libwindows*(S) library all send parameters in an *agentrect* structure. The agent command codes and their parameters are as follows:

A_NEWLAYER	followed by a two-byte channel number and a rectangle structure (four two-byte coordinates).
A_CURRENT	followed by a two-byte channel number.
A_DELETE	followed by a two-byte channel number.
A_TOP	followed by a two-byte channel number.
A_BOTTOM	followed by a two-byte channel number.
A_MOVE	followed by a two-byte channel number and a point to move to (two two-byte coordinates).
A_RESHAPE	followed by a two-byte channel number and the new rectangle (four two-byte coordinates).
A_NEW	followed by a two-byte channel number and a rectangle structure (four two-byte coordinates).

A_EXIT	no parameters needed.
A_ROMVERSION	no parameters needed. The response packet contains the size byte, two-byte return code, two unused bytes, and the parameter part of the terminal id string (e.g., "8;7;3").

Packets from the windowing terminal to Altos UNIX System V all take the following form:

command, data...

The single-byte commands are as follows:

C_SENDCHAR	Send the next byte to the UNIX system process.
C_NEW	Create a new UNIX system process group for this layer. Remember the window size parameters for this layer. The data for this command is in the form described by the *jwinsize* structure. The size of the window is specified by two 2-byte integers, sent low byte first.
C_UNBLK	Unblock transmission to this layer. There is no data for this command.
C_DELETE	Delete the UNIX system process group attached to this layer. There is no data for this command.
C_EXIT	Exit. Kill all UNIX system process groups associated with this terminal and terminate the session. There is no data for this command.
C_DEFUNCT	Layer program has died, send a terminate signal to the UNIX system process groups associated with this terminal. There is no data for this command.
C_SENDNCHARS	The rest of the data are characters to be passed to the UNIX system process.

C_RESHAPE The layer has been reshaped. Change the window size parameters for this layer. The data takes the same form as for the C_NEW command.

See Also

libwindows(S), jagent(M), xtproto(M), layers(C), xt(HW)

ld

invokes the link editor

Syntax

ld [options] filename

Description

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. It creates an executable program by combining one or more object files and copying the executable result to the file **a.out**. The *filename* must name an object or library file. By convention these names have the ".o" (for object) or ".a" (for archive library) extensions. If more than one name is given, the names must be separated by one or more spaces. If any input file, *filename*, is not an object file, *ld* assumes it is either an archive library or a text file containing link editor directives. By default, the file **a.out** is executable if no errors occurred during the load. If errors occur while linking, *ld* displays an error message; the resulting **a.out** file is unexecutable.

ld concatenates the contents of the given object files in the order given in the command line. Library files in the command line are examined only if there are unresolved external references encountered from previous object files.

The library is searched iteratively to satisfy as many references as possible and only those routines that define unresolved external references are concatenated. The library (archive) symbol table (see *ar*(F)) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there exist multiple library members defining the same external symbol. The library may be either a relocatable archive library or a shared library. Object and library files are processed at the point they are encountered in the argument list, so the order of files in the command line is important. In general, all object files should be given before library files. *ld* sets the entry point of the resulting program to the beginning of the first routine.

ld should be invoked using the *cc*(CP) command instead of invoking it directly. *cc* invokes *ld* as the last step of compilation, providing all the necessary C-language support routines. Invoking *ld* directly is not recommended since failure to give command line arguments in the correct order can result in errors.

Generating COFF vs. x.out Binaries

When **ld** is called, it scans all the object files that are to be linked. If they are all COFF objects, then the resulting binary will be in COFF format. If any of the object files to be linked are in *x.out* format, any COFF modules in the group will be converted to *x.out* and the resulting binary will be in *x.out* format.

Common Options

The following options are recognized by *ld*, and are common to producing both *COFF* and *x.out* binaries. Refer to the sections "Linking COFF Binaries" and "Linking x.out Binaries" for options specific to producing these binaries.

-o *name*
Sets the executable program filename to *name* instead of **a.out**.

-r XENIX VERSION: Invokes the incremental linker, **/lib/ldr**, with the arguments passed to **ld** to produce a relocatable output file.

AT&T VERSION: Retains relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor will not complain about unresolved references, and the output file will not be executable.

-s Strips line number entries and symbol table information from the output object file.

-u *symbol*
Designates the specified *symbol* as undefined. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the *ld* line is significant; it must be placed before the library which will define the symbol.

-V Outputs a message giving information about the version of *ld* being used.

Linking COFF Binaries

The following options are recognized by *ld* for linking *COFF* binaries:

-e *epsym*
Set the default entry point address for the output file to be that of the symbol *epsym*.

-f *fill*
Set the default fill pattern for "holes" within an output section as well as initialized *bss* sections. The argument *fill* is a two-byte constant.

-l*x* Search a library **lib***x***.a**, where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **-l** is significant. By default, libraries are located in *LIBDIR* or *LLIBDIR*.

-m
Produce a map or listing of the input/output sections on the standard output.

-a Create an absolute file. This is the default if the **-r** option is not used. Used with the **-r** option, **-a** allocates memory for common symbols.

-t Turn off the warning about multiply-defined symbols that are not the same size.

-x Do not preserve local symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

-z Do not bind anything to address zero. This option will allow run-time detection of null pointers.

-L *dir*
Change the algorithm of searching for **lib***x***.a** to look in *dir* before looking in *LIBDIR* and *LLIBDIR*. This option is effective only if it precedes the **-l** option on the command line.

-M
Output a message for each multiply-defined external definition.

-N Put the text section at the beginning of the text segment rather than after all header information, and put the data section immediately following text in the core image.

-VS *num*
Use *num* as a decimal version stamp identifying the **a.out** file that is produced. The version stamp is stored in the optional header.

-Y[*LU*],*dir*
Change the default directory used for finding libraries. If **L** is specified, the first default directory which *ld* searches, *LIBDIR*, is replaced by *dir*. If **U** is specified and *ld* has been built with a second default directory, *LLIBDIR*, then that directory is replaced by *dir*. If *ld* was built with only one default directory and **U** is specified a warning is printed and the option is ignored.

Linking x.out Binaries

The user must make sure that the most recent library versions have been processed with *ranlib*(CP) before linking. Library files for *x.out* format binaries must be in *ranlib*(CP) format, that is, the first member must be named __.SYMDEF, which is a dictionary for the library. *ld* compares the modification dates of the library and the __.SYMDEF entry, so if object files have been added to the library since __.SYMDEF was created, the link may result in an "invalid object module" that cannot run.

The following options are recognized by *ld* for linking *x.out* binaries:

-A *num*
Creates a standalone program whose expected load address (in hexadecimal) is *num*. This option sets the absolute flag in the header of the a.out file. Such program files can only be executed as standalone programs. Options **-A** and **-F** are mutually exclusive.

-B *num*
Sets the text selector bias to the specified hexadecimal number.

-c *num*
Alters the default target CPU in the *x.out* header. *num* can be 0, 1, 2, or 3 indicating 8086, 80186, 80286 and 80386 processors, respectively. The default on 8086/80286 systems is 0. The default on 80386 systems is 3. Note that this option only alters the default; if object modules containing code for a higher numbered processor are linked, then that will take precedence over the default.

-C
Causes the link editor to ignore the case of symbols.

-D *num*
Sets the data selector bias to the specified hexadecimal number.

-F *num*
Sets the size of the program stack to *num* bytes where num is a hexadecimal number. This option is ignored for 80386 programs which have a variable sized stack. By default 8086 programs have a variable stack located at the top of the first data segment, and 80286 programs have a fixed size 4096 byte stack. The **-F** option is incompatible with the **-A** option that cannot be opened by more than one user at the same time.

-g Includes symbolic information for *sdb*.

-i Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and shared by all users executing the file.

-La
Sets advisory file locking. Advisory locking is used on files with access modes that do not require mandatory locking.

-Lm
Sets mandatory file locking. Mandatory file locking is used on files that cannot be opened by more than one process at a time.

-m name
Creates a link map file named *name* that includes public symbols.

-M*x*
Specifies the memory model. *x* can have the following values:

s	small
m	middle
l	large
h	huge
e	mixed

-n *num*
Truncates symbols to the length specified by *num*.

-N *num*
Sets the pagesize to hex-*num* (which should be a multiple of 512) - the default is 1024 for 80386 programs. 8086/80186/80286 programs do not normally have page-aligned *x.out* files and the default for these is 0.

-P
Disables packing of segments

-R Ensures that the relocation table is of non-zero size. Important for 8086 compatibility.

-Rd num
Specify the data segment relocation offset (80386 only). *num* is hexadecimal.

-Rt num
Specify the text segment relocation offset (80386 only) *num* is hexadecimal.

-S *num*
Sets the maximum number of segments to *num*. If no argument is given, the default is 128.

Files

/bin/ld	
LIBDIR/libx.a	libraries
LLIBDIR/libx.a	libraries
a.out	output file
LIBDIR	usually /lib
LLIBDIR	usually /usr/lib

See Also

as(CP), cc(CP), masm(CP), mkshlib(CP), ranlib(CP), exit(S), end(S), a.out(F), ar(F).

Notes

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the program's address space.

- When the link editor is called through *cc*(CP), a startup routine is linked with the user's program. This routine calls exit( ) [see *exit*(S)] after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls exit( ) rather than falling through the end of the entry routine.

The symbols *etext*, *edata*, and *end* (see *end*(S)) are reserved and are defined by the link editor. It is incorrect for a user program to redefine them.

If the link editor does not recognize an input file as an object file or an archive file, it will assume that it contains link editor directives and will attempt to parse it. This will occasionally produce an error message complaining about "syntax errors".

Arithmetic expressions may only have one forward referenced symbol per expression.

If you are using XENIX binaries, please refer to the manual entry for this utility in the *XENIX Development Guide* for information on the appropriate usage with XENIX binaries.

Standards Conformance

ld is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

locale

the international locale

Syntax

language [_ [territory] [. [codeset]]]
"C"

Description

The international locale is a definition of the local conventions to be used by Altos UNIX System V libraries (and hence utilities and applications) for features whose behavior varies internationally.

The locale is specified by a character string of the form *language_territory.codeset*, where:

language represents both the language of text files being used, and the preferred language for messages (where the utility or application is capable of displaying messages in many languages),

territory represents the geographical location (usually the country) determining such factors as currency and numeric formats, and

codeset represents the character set in use for the internal representation of text.

The locale string "french_canada.8859" could therefore represent a Canadian user using the French language, processing data using the ISO 8859/1 standard international character set.

Each element (*language*, *territory* or *codeset*) can be up to 14 characters long, and should use only alphanumeric ASCII characters (see *ascii(M)*).

Note that the locale is not required to be completely specified: *territory* and *codeset* are optional. When a locale is incompletely specified, missing values are sought in the following sequence:

1. For each subclass, such as LC_TIME , in an environment variable of the same name as the subclass.

2. In the LANG environment variable.

3. In the file */etc/default/lang* .

The special locale string "C", used to represent the minimal environment needed for the C programming language, is taken to be equivalent to "english_us.ascii".

The format of the file */etc/default/lang* is at least one line, of the form:

LANG="*language_territory.codeset*"

A partly specified locale string will be expanded to the first *LANG* = entry in which the specified locale fields match.

Thus if the */etc/default/lang* file contains the following:

```
LANG=english_us.ascii
LANG=english_uk.8859
LANG=french_france.8859
```

A locale string "english_uk" will get expanded to "english_uk.8859", whereas a locale string "french" will get expanded to "french_france.8859".

The information used to configure a particular locale is generated by the utilities *chrtbl*(M), *coltbl*(M), *mestbl*(M), *montbl*(M), *numtbl*(M) and *timtbl*(M). The output files produced by these utilities (*ctype*, *collate*, *currency*, *messages*, *numeric* and *time* respectively) must be installed in the correct place in the directory structure */usr/lib/lang*. The correct directory name is found by substituting the language, territory and codeset names into the string "/usr/lib/lang/*language/territory/codeset*". The files should be installed into this directory with their existing file name (such as *ctype*).

A suggested naming convention for locales is as follows:

language The name of the language, in English, such as: english, french, german.

territory The name of the nation, in English, such as: us, uk, canada, france, germany, switzerland.

codeset An identification of the codeset, such as: ascii, 8859.

See Also

chrtbl(M), coltbl(M), environ(M), mestbl(M), montbl(M), numtbl(M), setlocale(S), timtbl(M)

Value Added

locale is an extension of AT&T System V provided in Altos UNIX System V.

log

interface to STREAMS error logging and event tracing

Description

log is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes [*strerr*(ADM), *strace*(ADM)]. *log* presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit *log* messages; and a subset of *ioctl*(S) system calls and STREAMS messages for interaction with a user level error logger, a trace logger, or processes that need to submit their own *log* messages.

Kernel Interface

log messages are generated within the kernel by calls to the function *strlog*:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in **<sys/strlog.h>** and **<sys/log.h>**. *mid* is the STREAMS module id number for the module or driver submitting the *log* message. *sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of SL_ERROR (the message is for the error logger), SL_TRACE (the message is for the tracer), SL_FATAL (advisory notification of a fatal error), and SL_NOTIFY (request that a copy of the message be mailed to the system administrator). *fmt* is a *printf(S)* style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to NLOGARGS (currently 3) numeric or character arguments can be provided.

User Interface

log is opened via the clone interface, **/dev/log**. Each open of **/dev/log** obtains a separate *stream* to *log*. In order to receive *log* messages, a process must first notify *log* whether it is an error logger or trace logger via a STREAMS I_STR *ioctl* call (see below). For the error logger, the I_STR *ioctl* has an ic_cmd field of I_ERRLOG with no accompanying data. For the trace logger, the *ioctl* has an ic_cmd field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct trace_ids elements. Each trace_ids structure specifies an *mid, sid,* and *level* from which message will be accepted. *strlog* will accept messages whose *mid* and *sid* exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of -1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

At most one trace logger and one error logger can be active at a time. Once the logger process has identified itself via the *ioctl* call, *log* will begin sending up messages subject to the restrictions noted above. These messages are obtained via the *getmsg(S)* system call. The control part of this message contains a log_ctl structure, which specifies the *mid, sid, level, flags*, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, and a sequence number. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of *log* messages can be determined.

Different sequence numbers are maintained for the error and trace logging *streams*, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic, some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to *log*, even if it is not an error or trace logger. The only fields of the log_ctl structure in the control part of the message that are accepted are the level and flags fields; all other fields are filled in by *log* before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to NLOGARGS) must be packed one word each, on the next word boundary following the end of the format string.

Attempting to issue an I_TRCLOG or I_ERRLOG when a logging process of the given type already exists will result in the error ENXIO being returned. Similarly, ENXIO is returned for I_TRCLOG *ioctls* without any trace_ids structures, or for any unrecognized I_STR *ioctl* calls. Incorrectly formatted *log* messages sent to the driver by a user process are silently ignored (no error results).

Examples

Example of I_ERRLOG notification:

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;              /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I_TRCLOG notification:

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;      /* any sub-id will be allowed */
tid[1].ti_level = -1;    /* any level will be allowed */

ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a *log* message (no arguments):

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

Files

/dev/log <sys/log.h> <sys/strlog.h>

See Also

strace(ADM), strerr(ADM), clone(M), intro(S), getmsg(S), putmsg(S)
STREAMS Programmer's Guide

login

gives access to the system

Syntax

login [name [env-var ...]]

Description

The *login* command is used at the beginning of each terminal session to identify the user and allow them access to the system. It cannot be invoked except when a connection is first established, or after the previous user has logged out by sending an end-of-file (Ctrl-D) to their initial shell.

login asks for your user name (if not supplied as an argument), and, if appropriate, your password and a dialup password. Echoing is turned off (where possible) during the typing of the passwords, so it will not appear on the written record of the session.

If you make a mistake in the login procedure you will receive the message

```
Login incorrect
```

and a new login prompt will appear. The number of login attempts you are allowed, is configurable. If you make too many unsuccessful login attempts, you or the terminal can be locked out.

If the login sequence is not completed successfully within a configurable period of time (e.g., one minute), the user is returned to the "login:" prompt or silently disconnected from a dial-in line.

After a successful login, accounting files (*/etc/utmp* and */etc/wtmp*) are updated, the user is notified if they have mail, and the start-up shell files (i.e., **.profile** for the Bourne shell or **.login** for the C-shell) if any, are executed.

login checks **/etc/default/login** for the following definitions of the the form DEFINE=value:

ALTSHELL
: If ALTSHELL is set to YES or if it is not present in **/etc/default/login**, then the SHELL environment variable is set to whatever shell is specified in the user's **/etc/passwd** entry. If ALTSHELL is set to NO, then the SHELL environment variable is set only if the shell is defined in the **/usr/lib/mkuser** directory (which

is list of recognized shells).

CONSOLE
The CONSOLE=*device* entry means that root can only log in on the device listed. For example, CONSOLE=/dev/console restricts root logins to the console device.

IDLEWEEKS
If a password has expired, the user is prompted to choose a new one. If it has expired beyond IDLEWEEKS, the user is not allowed to log in, and must consult system administrator. Works in conjunction with *passwd*(C). See cautions under **Notes**.

OVERRIDE
This allows root to log in on the console even if the protected password database entry for root is corrupted. *login* checks **/etc/default/login** to see if there is an entry similar to the following, which identifies the tty to be used when doing an override login for root:

```
OVERRIDE=tty01
```

PASSREQ
If PASSREQ=YES, this forces the user to select a password if they do not have one. PASSREQ=NO allows users to have accounts without passwords. See cautions under **Notes**.

SUPATH
If a user's UID is 0 (i.e. if this is the superuser), the PATH variable is set to SUPATH, if SUPATH is specified in **/etc/default/login.** It is not advisable for SUPATH to include the current directory symbol (.). Note that an empty directory ("::" or ":" at the beginning or end) is equivalent to ".".

ULIMIT
This variable defnes the maximum allowable file size. The default is 2,097,152 blocks, or 1 gigabyte. When setting ULIMIT, be sure to specify even numbers, as the ULIMIT variable accepts a number of 512-byte blocks.

UMASK
This is the default file creation mask (see *umask*(C)).

login initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(C)) according to specifications found in the **/etc/passwd** file. Argument 0 of the command interpreter is a dash (-) followed by the last component of the interpreter's pathname. The basic *environment* (see *environ*(M)) is initialized to:

HOME= your-login-directory
SHELL=*last field of passwd entry*
MAIL=/usr/spool/mail/*your-login-name*

Initially, *umask* is set to octal 022 by *login*.

Files

/etc/utmp	Information on current logins
/etc/wtmp	History of logins since last multiuser
/usr/spool/mail/*name*	Mailbox for user *name*
/etc/motd	Message of the day
/etc/default/login	Default values for environment variables
/etc/passwd	Password file
/etc/profile	System profile
$HOME/.profile	Personal profile

See Also

environ(M), getty(ADM), initscript(F), machine(HW), mail(C), newgrp(C), passwd(C), passwd(F), profile(M), su(C), sh(C), ulimit(S), umask(C), who(C)

Diagnostics

Not on system console
login is set up to allow root to log on on the console only, and you are not on the system console.

Login incorrect
The login or dialup password is incorrect.

Unable to change directory to dir
login cannot change directories to the home directory as specified by */etc/passwd.*

No utmp entry. You must exec 'login' from the lowest level 'sh'.
init did not put an entry in *utmp*.

No Root Directory
The shell field starts with a '*', and the attempt to do a *chroot* to the home directory failed.

You don't have a password.
A password is required and it hasn't been set previously.

Protected Password information suddenly vanished
During the course of working with the protected password database information the pointer pointing to the static version of the information has suddenly disappeared.

Cannot execute passwd program
The password program cannot be executed for some reason.

Login aborted due to no password.
The password program has returned an error while setting a password, as when the <DEL> key is pressed.

Can't rewrite protected password entry for user name,
Authentication error; see Account Administrator
The login program cannot update the protected password database entry.

Protected Password database problem
After updating Protected Password data, login reads the information again and the entry cannot be read.

Account is disabled but console login is allowed.
Account is disabled -- see Account Administrator.
If the account is locked, but root is logging in on the console (OVERRIDE tty), the first message is displayed; an ordinary user will see the second.

Account has been retired -- logins are no longer allowed.
The account is retired (no override for this condition).

Cannot set terminal mode.
The *chmod* of the tty failed.

Bad login user id.
No uid has been set. This can be due to a missing critical database file, such as */etc/auth/system/authorize*. Run *authck*(ADM) and check any error messages. This message will also be issued if login is run from an established login session rather that from *init*(M).

Wait for login retry.
Wait for login exit.
A login attempt has failed, and the system is configured to enforce a delay between login attempts.

user appears in /etc/passwd but not in Protected Password database
If the user is in */etc/passwd* but not in the Protected Password database, there is no message printed, but *login* generates the following audit record.

Cannot obtain database information on this terminal
login cannot get information from the */etc/auth/system/ttys* file for the tty line.

Error in terminal setup.
Something is wrong with the terminal setup (for example, stdin in, stdout, and stderr are the same thing).

Cannot obtain settings for this terminal
The *ioctl*(S) on the tty device failed.

No login program on root
When attempting to do a sublogin (chrooting to a subtree for a restricted login), no login program was found.

Can't rewrite terminal control entry for tty,
Authentication error; see Account Administrator
The information for the login tty cannot be updated.

Terminal Control information suddenly vanished
During the course of working with the terminal database information the pointer pointing to the static version of the information suddenly disappeared.

Bad priority setting.
nice failed to set the *nice* value specified in the Protected Password entry for the user.

Bad group id.
The call to *setgid* failed.

Bad user id.
The call to *setuid* failed.

Unable to set kernel authorizations.
The call to set the kernel authorizations failed.

Login timed out
login received an ALARM signal. Note: *login* sets this itself, but it could conceivably come from somewhere else.

Terminal is disabled but root login is allowed.
Terminal is disabled -- see Account Administrator.
If the terminal is disabled, but you are root login in on the console (OVERRIDE tty) the first message is displayed; the second is displayed for ordinary users.

The security databases are corrupt.
However, root login at terminal tty is allowed,
This is the message displayed when the OVERRIDE tty is used during a security problem.

Impossible to execute /bin/sh!
login cannot execute the shell program for doing an OVERRIDE.

Notes

login cannot be executed from a shell.

Environment variables such as HZ, PATH, and so forth should not be defined in **/etc/default/login**. Instead use */etc/initscript* to set global variables.

Sublogins (indicated by a shell of "*") are not supported and cause a warning.

Although IDLEWEEKS and PASSREQ are supported for compatibility with other UNIX systems, their use is not recommended. The proper way to set the behavior defined by these variables is by use of the *sysadmsh*(ADM) Accounts selection.

mapchan

configure tty device mapping

Syntax

mapchan [**-ans**] [**-f** *mapfile*] [*channels* ...]
mapchan [[**-o**] [**-d**]] [*channel*]

Description

mapchan configures the mapping of information input and output. The *mapchan* utility is intended for users of applications that employ languages other than English (character sets other than 7-bit ASCII).

mapchan translates codes sent by peripheral devices, such as terminals, to the internal character set used by the Altos UNIX System V system. *mapchan* can also map codes in the internal character set to other codes, for output to peripheral devices (such as terminals, printers, console screen, etc.). Note that PC keyboard configuration is accomplished through the *mapkey*(M) utility.

mapchan has several uses: to map a *channel* (**-a** or **-s**); to unmap a *channel* (**-n** and optionally **-a**); or to display the map on a channel (optionally **-o**, **-d**, *channels*).

mapchan with no options displays the map on the user's *channel*. The map displayed is suitable as input for *mapchan*.

The options are:

-a when used alone, sets all *channels* given in the default file (**/etc/default/mapchan**) with the specified map. When used with **-n**, it refers to all *channels* given in the default file. Superuser maps or unmaps all *channels*, other users map only *channels* they own. **-a** can not be used with **-d**, **-o**, or **-s**.

-d causes the mapping table currently in use on the given device, *channel*, to be displayed in decimal instead of the default hexadecimal. An ASCII version is displayed on standard output. This output is suitable as an input file to *mapchan* for another *channel*. Mapped values are displayed. Identical pairs are not output. **-d** can not be used with **-a**, **-f**, **-n**, **-o**, or **-s**.

-f causes the current *channel* or list of *channels* to be mapped with *mapfile*. **-f** can not be used with **-d**, **-n**, **-s**, or **-o**.

-n causes null mapping to be performed. All codes are input and output as received. Mapping is turned off for the user's *channel* or for other *channels*, if given. **-a** used with **-n** will turn mapping off on all *channels* given in the default file. This is the default mapping for all *channels* unless otherwise configured. **-n** can not be used with **-d**, **-f**, **-o**, or **-s**.

-o causes the mapping table currently in use on the given device, *channel*, to be displayed in octal instead of the default hexadecimal. An ASCII version is displayed on standard output. This output is suitable as an input file to *mapchan* for another port. Mapped values are displayed. Identical pairs are not output. **-o** can not be used with **-a**, **-d**, **-f**, **-n**, or **-s**.

-s sets the user's current *channel* with the *mapfile* given in the default file. **-s** can not be used with any other option.

The user must own the *channel* in order to map it. The super-user can map any channel. Read or write permission is required to display the map on a *channel*.

Each tty device *channel* (display adapter and video monitor on computer, parallel port, serial port, etc.) can have a different map. When Altos UNIX System V boots, mapping is off for all *channels*.

mapchan is usually invoked in the **/etc/rc2** scripts. These scripts are executed when the system enters multi-user mode and sets up the default mapping for the system. Users can invoke *mapchan* when they log in by including a *mapchan* command line in their **.profile** or **.login** file. In addition, users can remap their *channel* at any time by invoking *mapchan* from the command line. *channels* not listed in the default file are not automatically mapped. *channels* are not changed on logout. Whatever mapping was in place for the last user remains in effect for the next user, unless they modify their **.profile** or **.login** file.

For example, the default file **/etc/default/mapchan** can contain:

```
tty02      ibm
tty1a
tty2a      wy60.ger
lp         ibm
```

The default directory containing *mapfiles* is **/usr/lib/mapchan**. The default directory containing *channel* files is **/dev**. Full pathnames may be used for *channels* or *mapfiles*. If a *channel* has no entry, or the entry field is blank, no mapping is enabled on that *channel*. Additional *channels* added to the system, (for example, adding a serial or parallel port) are not automatically entered in the *mapchan* default file. If mapping is required, the system administrator must make the entries.

The format of the *mapfiles* is documented in the *mapchan*(F) manual page.

Using a Mapped channel

The input information is assumed to be 7- or 8-bit codes sent by the peripheral device. The device may make use of ''dead'' or ''compose'' keys to produce the codes. If the device does not have dead or compose keys, these keys can be simulated using *mapchan.*

One to one mapped characters are displayed when the key is pressed, and the mapped value is passed to the kernel.

Certain keys are designated as dead keys in the *mapfile*. Dead key sequences are two keystrokes that produce a single mapped value that is passed to the kernel. The dead key is usually a diacritical character, the second key is usually the letter being modified. For example, the sequence *´ e* could be mapped to the ASCII value 0xE9, and display as é.

One key is designated as the compose key in the *mapfile*. Compose key sequences are composed of three keystrokes that produce a single mapped value that is passed to the kernel. The compose key is usually a seldom used character or ctrl-*letter* combination. The second key is usually the letter being modified. The third key may be another character being combined, or a diacritical character. For example, if '@' is the compose key, the sequence @ *c O* could be mapped to the ASCII value 0xA9, and display as ©.

Characters are not echoed to the screen during a dead or compose sequence. The mapped character is echoed and passed to the kernel once the sequence is correctly completed.

Characters are always put through the input map, even when part of dead or compose sequences. The character is then checked for the internal value. The value may also be mapped on output. This should be kept in mind when preparing map files.

The following conditions will cause an error during input:

- non-recognized (not defined in the *mapfile*) dead or compose sequence
- restarting a compose sequence before completion by pressing the compose key in the middle of a dead or compose sequence. This is an error, but a new compose sequence is initiated.

If the *mapfile* contains the keyword *beep*, a bell sounds when either of the above conditions occurs. In either case, the characters are not echoed to the screen, or passed to the kernel.

In order to allow for character sequences sent to control the terminal (move the cursor, and so on) rather than to print characters on the screen, mapchan allows character sequences to be specified as special sequences which are not passed through the normal mapping procedure. Two sections may be specified, one for each of the input (keyboard) and output (screen) controls.

Character Sets

The internal character set used is defined by the *mapfiles* used. By default, this is the ISO 8859/1 character set which is also known as the dpANS X3.4.2 and ISO/TC97/SC2. It supports most of the Latin alphabet and can represent most European languages.

Several partial map files are provided as examples. They must be modified for use with specific peripheral devices. Consult your hardware manual for the codes needed to display the desired characters. Two map files are provided for use with the console device: **/usr/lib/mapchan/ibm** for systems with a standard PC character set ROM, and **/usr/lib/mapchan/iso** for systems with an optional ISO 8859/1 character set ROM.

Care should be taken that the *stty*(C) settings are correct for 8-bit terminals. The **/etc/gettydefs** file may require modification to allow logging in with the correct settings.

7-bit U.S. ASCII (ANSI X3.4) should be used if no mapping is enabled on the *channel*.

Files

/etc/default/mapchan
/usr/lib/mapchan/*

See Also

ascii(M), keyboard(HW), lp(C), lpadmin(ADM), mapchan(F), mapkey(M), parallel(HW), screen(HW), serial(HW), setkey(M), trchan(M), tty(M)

Notes

Some non-U.S. keyboards and display devices do not support characters commonly used by Altos UNIX System V command shells and the C programming language. It is not recommended that these devices be used for system administration tasks.

Printers can be mapped, output only, and can either be sent 8-bit codes or one-to-many character strings using *mapchan*. Line printer spooler interface scripts can be used (setuid *root*) to change the output map on the printer when different maps are required (as in changing print wheels to display a different character set). See *lp*(C) and *lpadmin*(ADM) for information on installing and administering interface scripts.

Not all terminals or printers can display all the characters that can be represented using this utility. Refer to the device's hardware manual for information on the capabilities of the peripheral device.

Warnings

Use of *mapfiles* that specify a different "internal" character set per-channel, or a set other than the 8-bit ISO 8859 set supplied by default can cause strange side effects. It is especially important to retain the 7-bit ASCII portion of the character set (see *ascii*(M)). Altos UNIX System V utilities and many applications assume these values.

Media transported between machines with different internal code set mappings may not be portable as no mapping is performed on block devices, such as tape and floppy drives. However, *trchan* with an appropriate *mapfile* can be used to "translate" from one internal character set to another.

Do not set ISTRIP (see *stty*(C)) when using *mapchan*. This option causes the eighth bit to be stripped before mapping occurs.

Value Added

mapchan is an extension of AT&T System V provided in Altos UNIX System V.

mapkey, mapscrn, mapstr, convkey

configure monitor screen mapping

Syntax

mapkey [**-dox**][*datafile*]
mapscrn [**-d**][*datafile*]
mapstr [**-d**][*datafile*]
convkey [**in** [**out**]]

Description

mapscrn configures the output mapping of the monitor screen on which it is invoked. *mapkey* and *mapstr* configure the mapping of the keyboard and string keys (eg. function keys) of the monitor (and multiscreens if present). *mapkey* can only be run by the super-user.

mapstr functions on a per-screen basis. Mapping strings on one screen does not affect any other screen.

If a file name is given on the argument line the respective mapping table is configured from the contents of the input file. If no file is given, the default files in **/usr/lib/keyboard** and **/usr/lib/console** is used. The **-d** option causes the mapping table to be read from the kernel instead of written and an ASCII version to be displayed on the standard output. The format of the output is suitable for input files to *mapscrn*, *mapkey*, or *mapstr* . Non-super-users can run *mapkey* and *mapstr* when the **-d** option is given.

With the **-o** or **-x** options, *mapkey* displays the mapping table in octal or hexadecimal.

convkey translates an old-style mapkey file into the current format. If *in* or *out* are missing, they default to *stdin* or *stdout.*

Files

/usr/lib/keyboard/*
/usr/lib/console/*

Notes

There is no way to specify that the map utilities read their configuration tables from standard input.

See Also

keyboard(HW), screen(HW), setkey(C)

Value Added

convkey, *mapkey*, *mapscrn* and *mapstr* are extensions of AT&T System V provided in Altos UNIX System V.

math

math functions and constants

Syntax

#include <math.h>

Description

This file contains declarations of all the functions in the Development System Math Library as well as various functions in the C Library that return floating-point values.

It defines the structure and constants used by the *matherr*(S) error-handling mechanisms, including the following constant used as an error-return value:

HUGE	The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (*e*).
M_LOG2E	The base-2 logarithm of *e*.
M_LOG10E	The base-10 logarithm of *e*.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	π, the ratio of the circumference of a circle to its diameter.
M_PI_2	$\pi/2$.
M_PI_4	$\pi/4$.
M_1_PI	$1/\pi$.
M_2_PI	$2/\pi$.
M_2_SQRTPI	$2/\sqrt{\pi}$.

M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

For the definitions of various machine-dependent "constants," see the description of the *<values.h>* header file.

See Also

intro(S), matherr(S), values(M)

Standards Conformance

math is conformant with:

The X/Open Portability Guide II of January 1987.

mconvert

compile menu into AOM

Syntax

mconvert input_file output_file

Description

The *mconvert* command is used for compiling a menu into an AOM menu file. The input_file is the path and file name of a form file, and the output_file is the path and file name of the compiled AOM menu file which can be

For more details see the Altos manual *AOM Tool Kit User's Guide*, part number 690-17464-*xxx*.

Files

/usr/aom/AOMcap
/usr/aom/aomtext
/usr/aom/aom.msgs
/usr/aom/defupdate
/usr/aom/form.format
/usr/aom/form.recover
/usr/aom/aomnames
/usr/aom/aomplanes
/usr/aom/creatext
/usr/aom/form.archive
/usr/aom/form.fsck
/usr/aom/setmsg

See Also

aom(M), fconvert(M)

messages

description of system console messages

Description

This section describes the various system messages which may appear on the system console. All messages are displayed in the following format:

label:*severity*:*comment*:*action*

The segments break down as follows:

label
Name of the driver or routine where the error occurred.

severity
The level of error severity, consisting of four levels:

PANIC	These fatal messages indicate hardware problems or kernel inconsistencies that are too severe for continued operation. After displaying a PANIC message, the system stops. Rebooting is required.
ERROR	Resource use has been affected. Some corrective action is needed.
WARNING	An error indication that should be monitored (example, free file space is low) but requires no immediate action.
INFO	Some information about the system is provided.

comment
A field containing information about the problem at hand.

action
The course of action to remedy the situation.

The system services error messages are generated by the shell and do not follow the above convention.

System Message Meanings

The following classifications are meant to be a key for you to use to determine the actions to take to correct an error situation. Each kernel message will have one of the following three classifications listed with it. The classifications are:

System inconsistency
A contradictory situation exists in the kernel.

Abnormal
A probably legitimate but extreme situation exists.

Hardware
Indicates a hardware problem.

System inconsistency messages indicate problems usually traceable to hardware malfunction, such as memory failure. These messages rarely occur since associated hardware problems are generally detected before such an inconsistency can occur.

Abnormal messages represent kernel operation problems, such as the overflow of critical tables. It takes extreme situations to bring these problems about, so they should never occur in normal system use. However, in some cases you can modify the kernel parameters that are causing the error message. Use the *configure*(ADM) utility to make the necessary changes.

Hardware messages normally specify the device, *dev*, that caused the error. Each message gives a device specification of the form *nn/mm* where *nn* is the major number of the device, and *mm* is its minor number. The command pipeline

ls -l /dev | grep *nn* **| grep** *mm*

may be used to list the name of the device associated with the given major and minor numbers.

System Messages

** Normal System Shutdown **
This message appears when the system has been shutdown properly. It indicates that the machine may now be rebooted or powered down.

kernel:PANIC:** ABNORMAL System Shutdown **
This message appears when errors occur during system shutdown. It is usually accompanied by other system messages. *System inconsistency, fatal.*

kernel:WARNING:bad block on dev *nn/mm*
A nonexistent disk block was found on, or is being inserted in, the structure's free list. *System inconsistency.*

kernel:WARNING:bad count on dev *nn/mm*
A structural inconsistency in the superblock of a file system. The system attempts a repair, but this message will probably be

followed by more complaints about this file system. *System inconsistency.*

kernel:WARNING:Bad free count on dev *nn/mm*

A structural inconsistency in the superblock of a file system. The system attempts a repair, but this message will probably be followed by more complaints about this file system. *System inconsistency.*

kernel:ERROR:error on dev *name* (*nn/mm*)

This is the way that most device driver diagnostic messages start. The message will indicate the specific driver and complaint. The *name* is a word identifying the device.

kernel:ERROR:iaddress > 2^24

This indicates an attempted reference to an illegal block number, one so large that it could only occur on a file system larger than 8 billion bytes. *Abnormal.*

kernel:WARNING:Inode table overflow

Each open file requires an inode entry to be kept in memory. When this table overflows, the specific request (usually *open*(S) or *creat*(S)) is refused. Although not fatal to the system, this event may damage the operation of various spoolers, daemons, the mailer, and other important utilities. Abnormal results and missing data files are a common result. Use *configure*(ADM) to raise the number of inodes. *Abnormal.*

kernel:WARNING:interrupt from unknown device, vec=*num*

The CPU received an interrupt via a supposedly unused vector. This message is followed by "panic:unknown interrupt." Typically, this event comes about when a hardware failure miscomputes the vector of a valid interrupt. *Hardware.*

kernel:WARNING:stray interrupt on vector *num*

The CPU received an interrupt via a supposedly unused vector. *Hardware.*

kernel:WARNING:no file

There are too many open files. The system has run out of entries in its "open file" table. The warnings given for the message "inode table overflow" apply here. Use *configure*(ADM) to raise the total number of available files or the number of files available per process. *Abnormal.*

kernel:WARNING:no space on dev *nn/mm*

This message means that the specified file system has run out of free blocks. Although not normally as serious, the warnings discussed for "inode table overflow" apply:often user programs are written casually and ignore the error code returned when they tried to write to the disk; this results in missing data and "holes" in data files. The system administrator should keep close watch on the amount of free disk space and take steps to avoid this situation. *Abnormal.*

kernel:WARNING:Out of inodes on dev *nn/mm*

The indicated file system has run out of free inodes. The number of inodes available on a file system is determined when the file system is created (using *mkfs*(ADM)). The default number is quite generous; this message should be very rare. The only recourse is to remove some worthless files from that file system, or dump the entire system to a backup device, run *mkfs*(ADM) with more inodes specified, and restore the files from backup. *Abnormal.*

kernel:PANIC:blkdev

An internal disk I/O request, already verified as valid, is discovered to be referring to a nonexistent disk. *System inconsistency, fatal.*

kernel:PANIC:devtab

An internal disk I/O request, already verified as valid, is discovered to be refering to a nonexistent disk. *System inconsistency, fatal.*

kernel:PANIC:iinit

The super-block of the root file system could not be read. This message occurs only at boot time. *Hardware, fatal.*

kernel:PANIC:swap IO error

A fatal I/O error occurred while reading or writing the swap area. *System inconsistency, fatal.*

kernel:PANIC:memory failure - parity error

A hardware memory failure trap has been taken. *System inconsistency, fatal.*

kernel:PANIC:no fs

A mounted file system's entry has disappeared from the system mount table. *System inconsistency, fatal.*

kernel:PANIC:no imt

A mounted file system has disappeared from the mount table. *System inconsistency, fatal.*

kernel:PANIC:no procs

Each user is limited in the amount of simultaneous processes he can have; an attempt to create a new process when none is available or when the user's limit is exceeded and refused. That is an occasional event and produces no console messages; this panic occurs when the kernel has certified that a free process table entry is available and can't find one when it goes to get it. *System inconsistency, fatal.*

kernel:WARNING:Out of swap

There is insufficient space on the swap disk to hold a task. The system refuses to create tasks when it feels there is insufficient disk space, but it is possible to create situations to circumvent this mechanism. *Abnormal.*

kernel:PANIC:general protection trap

General protection trap taken in kernel. *System inconsistency, fatal.*

kernel:PANIC:segment not present

An attempt has been made to access an invalid segment. It may also indicate the segment-not-present trap has been taken in the kernel. *System inconsistency, fatal.*

kernel:PANIC:Timeout table overflow

The timeout table is full. Timeout requests are generated by device drivers, there should usually be room for one entry per system serial line plus ten more for other usages. Use *configure*(ADM) to raise the number of timeout table entries.

kernel:PANIC:Trap in system

The CPU has generated an illegal instruction trap while executing kernel or device driver code. This message is preceded with an information dump describing the trap. *System inconsistency, fatal.*

kernel:PANIC:Invalid TSS

Internal tables have become corrupted. *System inconsistency, fatal.*

kernel:WARNING:bootstring invalid, ignored

A bad bootstring was entered at the Boot prompt.

kernel:ERROR:bad syntax - *string*

A bad bootstring was entered at the Boot prompt.

kernel:PANIC:bad mapping in copyio

Copyio was called with a strange request. Usually a bad driver.

kernel:WARNING:HARDWARE FAILURE:386 incorrectly multiplies 32-bit numbers

The cpu is displaying the 32-bit multiply bug.

kernel:PANIC:*** POWER CYCLE TO REBOOT ***

This message follows the above HARDWARE FAILURE 32 bit error message.

kernel:INFO:10 bits of I/O address decoding

The hardware is only decoding 10 bits of i/o addresses. This amount is sufficient in most cases. This condition is only an issue if you are strapping i/o devices with a base address above

400 (hex).

kernel:WARNING:A31 CPU bug workaround not possible for this machine
A31 was specified on the boot line, but cannot be applied to the current system.

kernel:INFO:A31 CPU bug workaround in effect
A31 was specified on the boot line and the software workaround is currently in effect.

kernel:PANIC:bad boot string An invalid boot string was entered at the Boot prompt.

kernel:PANIC:out of both memory & swap
No more memory pages or swap pages are free.

kernel:PANIC:not enough contiguous memory
The kernel memory allocation routines require more physically contiguous memory. Either decrease the size of some kernel parameters (like disk buffers) or add more physical memory.

kernel:WARNING:filesystem page read failed
An error occurred trying to read a page from the disk. This is not fatal, but usually indicates hardware problems.

kernel:PANIC:free inode isn't
There is internal inode table corruption within the kernel.

kernel:ERROR:Map overflow (*num*), shutdown and reboot, mp->mpent
There are internal kernel map inconsistencies. Reboot your system.

kernel:PANIC:write_sb():cannot cvts3superb() yet
This message is found in the 386 kernel only. A write of a non SYS III or SYS V filesystem superblock is being attempted. This action should be impossible due to earlier checks.

kernel:WARNING:Can't allocate message buffer
This message indicates a lack of memory. Processes should be killed to make more room. Another option is to add more physical memory.

kernel:PANIC:Large model 386 ssig
Internal kernel error in processing large model 386 signals.

Trap *type*
This message precedes a "kernel:PANIC:" message. The *type* is the trap number given by the processor. The message is followed by a dump of registers. *System inconsistency, fatal.*

fpsave:PANIC:no fp_task
No floating point context to save, internal kernel error.

mdep.386/fp.c:WARNING:No floating point emulator found in *string*,
No */etc/emulator* was present in the root filesystem. The System Administrator should install one and reboot.

fp_OVERRUN:PANIC:coprocessor overrun - with no 287/387
Internal coprocessor error. fatal.

fp_COPROC:PANIC:, coprocessor error - with no 287/387
Inconsistent kernel internal state.

fp_COPROC:PANIC:coprocessor error - switched away from fp_task
Internal kernel mismanagement of floating point processes.

fp_DNA:PANIC:
A device trap happened while emulating floating point instructions.

iinit:PANIC:cannot copy in superblock
An error happened during the root filesystem superblock loading.

srmount:PANIC:cannot cvtv7superb() yet
A root filesystem superblock was not recognized as a SYS III or SYS V superblock. V7 superblocks cannot currently be converted on the 386 kernel.

mapphys:PANIC:sptmap overflow
No system page table pages are available. This is an internal error in the kernel, usually caused by a faulty device driver.

physio:PANIC:bad state A device driver made an invalid request to physio.

badint:PANIC:bad interrupt handler Invalid interrupt request, usually fault hardware.

setup:PANIC:sptmap overflow This message indicates possible kernel image corruption or lack of physical memory.

setup:PANIC:u-area not page aligned This indicates possible kernel image corruption.

setup:PANIC:u-area address does not match SPTADDR
Indicates possible kernel image corruption.

cmn_err:PANIC:DOUBLE PANIC The kernel panicked while trying to panic. You must power cycle at this point to reboot the machine.

cmn_err:PANIC:unknown level in cmn_err (level=*num*, msg=*string*),
The kernel's cmn_err() routine was called with an invalid argument.

Kernel Paging Messages

The following messages indicate system inconsistencies in the kernel paging code. These inconsistencies can be caused by hardware or software problems. Reboot your system and note the circumstances if you see one of these messages:

mfalloc:PANIC:page not free

mfalloc:PANIC:page not free at exit

mffree:PANIC:page already free

mffree:PANIC:page is locked

dfalloc:PANIC:frame not free at exit

xlcheck:PANIC:xlink serial mismatch

impcode:PANIC:called to load impure 386

impcode:PANIC:more than 1 data segment?

preload:PANIC:, invalid page (*num*, *num*)

kernel:PANIC:bad page type for protection fault

kernel:PANIC:protection fault on read access

kernel:PANIC:not present fault on shared data

kernel:PANIC:added strange page table - *num*, index

pgfind:PANIC:not in cache

pghash:PANIC:not in cache

pginval:PANIC:list broken

pginval:PANIC:not in cache

mftomp:PANIC:bad frameno *num*

mptomf:PANIC:bad mp *num*

swapadd:PANIC:no space for dpfi

dftodp:PANIC:bad frameno *num*

dptodf:PANIC:bad dp *num*

dptodf:PANIC:bad dp *num*

pgread:PANIC:no xlink

pgfree:PANIC:invalid page marked present

pgfree:PANIC:freeing intransit page

pgpid:WARNING:setting disk pid

kernel:PANIC:page table under page table?

kernel:PANIC:swapping intransit page

dftomf:PANIC:non-swap page table entry changed

dftomf:PANIC:swap disk frame rcnt(*num*) != 1, dp=*num*, dp->dp_rcnt,dp

dftomf:PANIC:page type mismatch - mptype *num* dptype *num* mp *num* dp *num*, mp->mp_type, dp->dp_type, mp, dp

dftomf2:PANIC:, swap memory frame rcnt(*num*) != 1, mp=*num*,

dftomf3:PANIC:swap mem frame rcnt(*num*) != 1, mp=*num*, mp->mp_rcnt, mp

mftodf1:PANIC:swap mem frame rcnt(*num*) != 1, mp=*num*, mp->mp_rcnt, mp

mftodf:PANIC:memory frame marked in transit

mftodf:PANIC:page type mismatch - dptype *num* mptype *num* dp *num* mp *num*

mftodf2:PANIC:swap disk frame rcnt(*num*) != 1, dp=*num*

mftodf3:PANIC:swap disk frame rcnt(*num*) != 1, dp=*num*, dp->dp_rcnt, dp fftomf:PANIC:page type(*num*) not TE_FILSYS, mp = *num*,mp->mp_type, mp

mfcvt:PANIC:zero ref count

ptdup:PANIC:TE_SWAP page rcnt(*num*) > 1,

ptdup:PANIC:xlinked page has reference

ptdup2:PANIC:TE_SWAP page rcnt > 1

ptdup:PANIC:xlinked page has reference

ptdup:PANIC:locked page not present

ptdup:PANIC:intransit page

pgcheck:PANIC:page type mismatch:ptp *num* type *num* xtype *num*,ptp,type,xtype

The above listed messages indicate system inconsistencies in the kernel paging code. These inconsistencies can be caused both by hardware or software problems. Reboot your system.

cputok:PANIC:

cpktou:PANIC:

sdfrcm:PANIC:sdp->sd_inode not found

The above 3 errors indicate internal shared data errors within the kernel.

v86sighdlint:WARNING:lost signal

v86setint:PANIC:xtss pte not present

The above 2 errors indicate internal VPIX processing errors within the kernel.

namei:PANIC:null cache ino

namei:PANIC:duplicating cache

The above 2 messages indicate internal file management errors in the kernel.

System Services Messages

The following messages are displayed by the shell when a system call fails.

Not owner:
Typically, this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

No such file or directory:
This error occurs when a filename is specified and the file should exist but doesn't, or when one of the directories in a pathname does not exist.

No such process:
No process can be found corresponding to that specified by *pid* in *kill* or *ptrace*.

Interrupted system call:
An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

I/O error:
Some physical I/O error. This error may in some cases occur on a call following the one to which it actually applies.

No such device or address:
I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.

Arg list too long:
An argument list longer than 5,120 bytes is presented to a member of the *exec* family.

Exec format error:
A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see *a.out*(F)).

Bad file number:
Either a file descriptor refers to no open file, or a read (respectively write) request is made to a file which is open only for writing (respectively reading).

No child processes:
A *wait* was executed by a process that had no existing or unwaited-for child processes.

No more processes:
A *fork* failed because the system's process table is full or the user is not allowed to create any more processes.

Not enough space:
During an *exec*, or *sbrk*, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a *fork*.

Permission denied:
An attempt was made to access a file in a way forbidden by the protection system.

Bad address:
The system encountered a hardware fault in attempting to use an argument of a system call.

Block device required:
A nonblock file was mentioned where a block device was required, e.g., in *mount*.

Device busy:
An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled.

File exists:
An existing file was mentioned in an inappropriate context, e.g., *link*.

Cross-device link:
A link to a file on another device was attempted.

No such device:
An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.

Not a directory:
A nondirectory was specified where a directory is required, for example, in a path prefix or as an argument to *chdir*(S).

Is a directory:
An attempt to write on a directory.

Invalid argument:
An invalid argument (e.g., dismounting a nonmounted device; mentioning an undefined signal in *signal* or *kill*; reading or writing a file for which *lseek* has generated a negative pointer). Also set by the math functions described in the (S) entries of this manual.

File table overflow:
The system's table of open files is full and temporarily no more *opens* can be accepted.

Too many open files:
No process may have more than 60 file descriptors open at a time.

Not a character device

Text file busy:
An attempt to execute a pure-procedure program which is currently open for writing (or reading). Also an attempt to open for writing a pure-procedure program that is being executed.

File too large:
The size of a file exceeded the maximum file size (1,082,201,088 bytes) or ULIMIT; see *ulimit*(S).

No space left on device:
During a *write* to an ordinary file, there is no free space left on the device.

Illegal seek:
An *lseek* was issued to a pipe.

Read-only file system:
An attempt to modify a file or directory was made on a device mounted read-only.

Too many links:
An attempt to make more than the maximum number of links (1000) to a file.

Broken pipe:
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

Arg out of domain of func:
The argument of a function in the math package is out of the domain of the function.

Result too large:
The value of a function in the math package is not representable within machine precision.

File system needs cleaning:
An attempt was made to *mount*(S) a file system whose super-block is not flagged clean.

Would deadlock:
A process' attempt to lock a file region would cause a deadlock between processes vying for control of that region.

Not a name file:
A *creatsem*(S), *opensem*(S), *waitsem*(S), or *sigsem*(S) was issued using an invalid semaphore identifier.

Not available:
An *opensem*(S), *waitsem*(S) or *sigsem*(S) was issued to a semaphore that has not been initialized by a call to *creatsem*(S). A *sigsem* was issued to a semaphore out of sequence; i.e., before the process has issued the corresponding *waitsem* to the semaphore. An *nbwaitsem* was issued to a semaphore guarding a resource that is currently in use by another process. The semaphore on which a process was waiting has been left in an inconsistent state when the process controlling the semaphore exits without relinquishing control properly; i.e., without issuing a *waitsem* on the semaphore.

A name file:
A name file (semaphore, shared data, etc.) was specified when not expected.

No message of desired type: An attempt was made to receive a message of a type that does not exist on the specified message queue [see *msgop*(S)].

Identifier removed:
This error is returned to a process that resumes execution due to the removal of an identifier from the file system's
name space; see *msgctl*(S), *semctl*(S), and *shmctl*(S).

No record locks available:
In *fcntl*(S) the setting or removing of record locks on a file cannot be accomplished because there are no more record entries left on the system.

Channel number out of range

Level 2 not synchronized

Level 3 halted

Level 3 reset

Link number out of range

Protocol driver not attached

No CSI structure available

Level 2 halted

Deadlock situation detected/avoided
A deadlock situation was detected and avoided. This error pertains to file and record locking.

No record locks available

Bad exchange descriptor

Bad request descriptor

Message tables full

Inode table overflow

Bad request code

Invalid slot

File locking deadlock

Bad font file format

Not a stream device
A *putmsg*(S) or *getmsg*(S) system call was attempted on a file descriptor that is not a STREAMS device.

No data available

Timer expired
The timer set for a STREAMS *ioctl*(S) call has expired. The cause of this error is device specific and could indicate either a hardware or software failure, or perhaps a timeout value that is too short for the specific operation. The status of the *ioctl*(S) operation is indeterminate.

Out of stream resources
During a STREAMS *open*(S), either no STREAMS queues or no STREAMS head data structures were available.

Machine is not on the network
This error is Remote File Sharing (RFS) specific. It occurs when users try to advertise, unadvertise, mount, or unmount remote resources while the machine has not done the proper startup to connect to the network.

Package not installed
This error occurs when users attempt to use a system call from a package which has not been installed.

Object is remote
This error is RFS specific. It occurs when users try to advertise a resource which is not on the local machine, or try to mount/unmount a device (or pathname) that is on a remote machine.

Link has been severed
This error is RFS specific. It occurs when the link (virtual circuit) connecting to a remote machine is gone.

Advertise error
This error is RFS specific. It occurs when users try to advertise a resource which has been advertised already, or try to stop the RFS while there are resources still advertised, or try to force unmount a resource when it is still advertised.

Srmount error
This error is RFS specific. It occurs when users try to stop RFS while there are resources still mounted by remote machines.

Communication error on send
This error is RFS specific. It occurs when trying to send messages to remote machines but no virtual circuit can be found.

Protocol error
Some protocol error occurred. This error is device specific, but is generally not related to a hardware failure.

Multihop attempted
This error is RFS specific. It occurs when users try to access remote resources which are not directly accessible.

Not a data message
During a *read*(S), *getmsg*(S), or *ioctl*(S) I_RECVFD system call to a STREAMS device, something has come to the head of the queue that can't be processed. That something depends on the system call:
read(S) - control information or a passed file descriptor.
getmsg(S) - passed file descriptor.
ioctl(S) - control or data information.

Name not unique on network

File descriptor in bad state

Remote address changed

Cannot access a needed shared library
Trying to *exec*(S) an *a.out* that requires a shared library (to be linked in) and the shared library doesn't exist or the user doesn't have permission to use it.

Accessing a corrupted shared library
Trying to *exec*(S) an *a.out* that requires a shared library (to be linked in) and *exec*(S) could not load the shared library. The shared library is probably corrupted.

Trying to *exec*(S) an *a.out* that requires a shared library (to be linked in) and there was erroneous data in the .lib section of the *a.out*. The .lib section tells *exec*(S) what shared libraries are needed. The *a.out* is probably corrupted.

Attempting to link in more shared libraries than system limit
Trying to *exec*(S) an *a.out* that requires more shared libraries (to be linked in) than is allowed on the current configuration of the system. See the System Administrator's Guide.

Cannot exec a shared library directly
Trying to *exec*(S) a shared library directly. This is not allowed.

Driver Messages

The following messages are different from kernel messages in that they are generated by the device drivers for the various hardware supported under Altos UNIX System V. The source of the message can be determined by checking the *label* field of the message.

Console Driver Messages

console:WARNING:Kernel messages lost on non-text screen
(also check /usr/adm/messages)
Kernel messages were lost while the console was in graphics mode and did not appear. Check the last lines of **/usr/adm/messages** to find the messages.

console:WARNING:Too many keyboard groups
There are more video devices attached to your system than your kernel is designed to support.

Irwin Driver Messages

IRWIN:ERROR:Tape bad block table was not successfully read.
When the tape device is open the bad block table is read into memory. This messages indicates that the read did not work correctly.

IRWIN:ERROR:Tape is not formatted.
The tape must be formatted before use.

IRWIN:ERROR:Tape is write protected.
The write protect tab must be removed for use.

IRWIN:ERROR:Cannot write to DC1000 cartridge.
Only Irwin model 110 or 125 drives can write to DC1000 cartridges.

IRWIN:ERROR:Not enough memory for mini-cartridge; retrying...
The Irwin is waiting for enough user memory to become available to use the device.

IRWIN:ERROR:Not enough memory for mini-cartridge; open failed.
The Irwin did not get enough memory to be able to use the device after several retries.

IRWIN:ERROR:Tape write error.
A write attempt was unsuccessful for an unknown reason.

IRWIN:ERROR:Tape verify error.
A verify attempt was unsuccessful for an unknown reason.

IRWIN:ERROR:Tape read error.
A read attempt was unsuccessful for an unknown reason.

IRWIN:ERROR:Tape uncorrectable ECC error.
An uncorrectable ECC memory error has occurred, check your hardware for defective chips.

IRWIN:ERROR:Cannot format DC1000 cartridge.
Only Irwin model 110 or 125 drives can write to DC1000 cartridges.

IRWIN:ERROR:Bad state:*num*
Unknown state in the interrupt routine.

IRWIN:ERROR:DMA boundary error - start address:*num* ending address:*num*
Device tried to transfer data from a buffer that crosses a 64k boundary.

Cartridge Driver Messages

CT:ERROR:Tape controller (type=*name*) not found
The controller specified in in the file */usr/sys/io/ctconf.asm* was not found.

CT:ERROR:Cartridge tape is write protected
You must remove the write protect tab from the cartridge before use.

CT:ERROR:system too busy for efficient tape use
: There is not enough user memory available to allow the device to work.

CT:WARNING:attempted to free invalid buffer
: The driver attempted free a buffer that was not active. The buffer must be activated before use.

SCSI Driver Messages

scsi:ERROR:No controller response :*num*
: Requested controller is not present on SCSI bus *num*. Check your system setup and connections.

scsi:ERROR:CTLR *num* LUN *num* not attached
: Requested unit not present on controller. Check your system setup.

scsi:ERROR:CTLR *num* LUN *num*:invalid type <*num*>,
: Requested unit is not a disk or tape. Disk and tape and printer are currently the only supported SCSI devices.

scsi:ERROR:CTLR *num* LUN *num*:device not ready, ctlr, x);
: Requested device is busy.

scsi:ERROR:adstrategy:device/type error 0x*type*/0x*type*
: Internal error - open device is not disk, tape or printer.

scsi:ERROR:adioctl:ADMODESENSE rc *num* host *num* unit *num*
: ioctl sense command did not complete as expected.

scsi:WARNING:adioctl:ADEXECUTE rc *num* host *num* unit *num*
: ioctl execute command did not complete as expected.

scsi:INFO:adioctl:*num* reassigned
: ioctl bad block mapping completed (done in pairs)

scsi:WARNING:adsetparam:ADMODESENSE rc *num* host *num* unit *num*
: Mode sense command did not complete as expected.

scsi:ERROR:adgetcdb:unsupported command *num*
: Internal error - unexpected command.

scsi:WARNING:adintr:adapter *num* SR_DETECTED status=*num*, intr=*num*
: SCSI reset detected.

scsi:WARNING:Unexpected MBI status *num*
: Unexpected condition after interrupt.

scsi:WARNING:ad_sndcmd:unexpected port status = *num*
Unable to send command to adapter.

scsi:ERROR:adpresent:Adapter *num* internal failure:*num*
Adapter returned bad status on initialization.

scsi:ERROR:on disk dev=*num/num* ha=*num* id=*num* lun=*num* block=*num* sector=*num*, cylinder/head = *num/num*
Disk I/O failure.

scsi:ERROR:on tape ha=*num* id=*num* lun=*num* hst *num* ust *num*
AHA-1540 cmd :*num* [*num* ...]
AHA-1540 sense :*num* [*num* ...]
Tape I/O failure; followed by one of these messages:

end of tape
tape is write protected
wrong record length

Disk Driver Messages

disk:ERROR:Diskinfo table overflow
Too many disk drives in use - reconfigure kernel to increase the available number of disks.

disk:ERROR:Invalid partition sector on hard disk
Master boot block on disk is unrecognizable. Run fsck(ADM).

Floppy Driver Messages

floppy:WARNING:CMOS indicates no diskette drives installed
Configuration memory invalid - run your DOS SETUP disk.

floppy:WARNING:CMOS indicates diskette drive *num* not present
Configuration memory invalid - run your DOS SETUP disk.

floppy:ERROR:fd*num* being formatted
The floppy drive is in use.

floppy:ERROR:disk is write protected
The disk cannot be written because it is protected.

floppy:ERROR:on dev (*num/num*), block=*num* cmd=*num* status=*num*
Floppy I/O failure. possibly followed by the message:
insert disk or close floppy door
if appropriate.

floppy:WARNING:cmd result error
I/O error on the floppy drive.

VPIX Messages

VPIX:command completed unexpectedly
Process terminated prematurely.

OMTI Driver Messages

omti:ERROR:cannot allocate a GDT descriptor
Internal error - kernel dscralloc routine failed.

omti:ERROR:unit=*num* controller not configured
Internal error - driver open failed to identify disk type.

omti:WARNING:already busy
Internal error - omtistart called for a busy drive.

omti:ERROR:unknown command(*num*), bp->b_cmd
Internal error - omtistart encountered an unrecognized command.

omti:ERROR:command setup failed
Controller failed to accept command.

omti:WARNING:non-omti interrupt (*num*), omti_status
Controller did not signal an interrupt when an interrupt was received.

omti:WARNING:unexpected omti interrupt (*num*), omti_status
Internal error - no pending command when interrupt received.

omti:WARNING:still busy
Controller still busy after generating an interrupt.

omti:ERROR:during omti_sense
Interrupt received during an OMTI sense command.

omti:ERROR:initialization failure
Error indicated during an initialization.

omti:ERROR:sense command setup failed
Controller failed to accept setup command.

omti:ERROR:minor=*num*, block=*num*, errtype=*num*, code=*num*, unit=*num* [sector=*num*, cylinder/head=*num*/*num*,] <message>
Disk I/O failure. <message> is one of:

No error or no sense information,
No Index,
No Seek/Command Complete,
Write/Drive Fault,
Drive Not Selected/Not Ready,

No Track zero or Cylinder zero found,
Multiple Drives Selected,
Seek/Command in progress,
Cartridge Changed
ID CRC,
Uncorrectable Data ECC,
ID Address Mark Not Found,
Data Address Mark Not Found,
Sector Not Found,
Seek Error,
Sequence/DMA,
Write Protected,
Correctable ECC,
Bad Track Encountered,
Illegal Interleave Factor,
Unknown Error,
Ilegal Access To An Alternated Track/Unable to Read the Alternate Track Address,
Alternate of Bad Track Already Assigned,
No Alternate Track Found,
Illegal Alternate Track Address
Invalid Command,
Illegal Disk Address,
Illegal Function for Drive Type,
Volume Overflow
RAM error,
EPROM Checksum/Internal Diagnostic error
Error with unknown type or code

omti:ERROR:controller already in select state
  Internal error - controller busy when sending command.

omti:ERROR:cannot enter command phase
  Controller failed to accept select command.

omti:ERROR:C_D bit stuck off
  Controller failed to indicate readiness for command.

omti:ERROR:OMTI_BUSY bit still stuck on
  Controller failed to obey reset command.

omti:INFO:unloading all requests
  Preparing for manual reset because programmed reset did not work.

omti:WARNING:colliding polling routines ...
  Internal error - multiple instances of omtipoll.

omti:ERROR:timed out
  Expected interrupt did not arrive.

omti:ERROR:please use sfmt to modify disk parameters
Attempt to write disk characteristics directly with DIOWDISK ioctl.

Serial Driver Messages

serial:ERROR:Garbage or loose cable on dev *num*, port shut down
Too many interrupts were received together. Check your connections.

Winchester Driver Messages

wd:ERROR:on fixed disk dev=*num*/*num* block=*num* cmd=*num* status=*num* sector=*num*, cylinder/head = *num*/*num*
Disk I/O failure.

Event Driver Messages

event:ERROR:event channel full
There are no more devices available in the event queue.

event:ERROR:event table full
All of the system's event queues are opened.

Keyboard Driver Messages

kb:ERROR:keyboard is in an unknown mode
The keyboard has been set in an invalid mode through an *ioctl*().
The only valid keyboard modes are XT (0) and AT(1).

Notes

Not all messages appear on all machines. Some messages are processor dependent.

mestbl

create a messages locale table

Syntax

mestbl [*specfile*]

Description

The utility *mestbl* is provided to allow *LC_MESSAGES* locales to be defined. It reads in a specification file (or standard input if *specfile* is not defined), containing a definition for a particular locale's response strings to yes/no queries, and produces a concise format table file, to be read by *setlocale*(S).

The response strings may be specified as a string held within double quotes or as a series of characters which are specified in one of six different ways (the following examples all specify the ASCII character 'A'):

```
65      - decimal
0101    - octal
0x41    - hexadecimal
'A'     - quoted character
'\101'  - quoted octal
'\x41'  - quoted hexadecimal
```

or a combination of both methods, for example:

```
'y' "es"
```

is identical to:

```
"yes"
```

To specify the response strings, the above string definitions must be preceded by the keyword **YESSTR=** for affirmative responses, and **NOSTR=** for negative responses.

All characters following the hash character are treated as a comment and ignored up to the end of the line, unless the hash is within a quoted string.

The concise format locale table is placed in a file named *messages* in the current directory. This file should be copied or moved to the correct place in the *setlocale* (S) file tree (see *locale* (M)). To prevent accidental corruption of the output data, the file is created with no write permission; if the mestbl utility is run in a directory containing a write-protected "messages" file, the utility will ask if the existing file should be replaced - any response other than "yes" or "y" will cause mestbl to terminate without overwriting the existing file.

See Also

chrtbl(M), montbl(M), coltbl(M), locale(M), numtbl(M), timtbl(M), setlocale(S)

Diagnostics

All error messages printed are self explanatory.

Value Added

mestbl is an extension of AT&T System V provided in Altos UNIX System V.

montbl

create a currency locale table

Syntax

montbl [*specfile*]

Description

The utility *montbl* is provided to allow new LC_MONETARY locales to be defined; it reads a specification file, containing a definition of the currency symbol for a particular locale, and produces a binary table file, to be read by *setlocale* (S), which determines the behavior of the *nl_langinfo* (S) routine.

The information supplied in the specification file consists of a line in the following format:

CRNCYSTR = *string*

The " = " can be separated from the keyword and string fields by zero or more space or tab characters.

The *string* is a sequence of characters surrounded by quotes ("). The first character of the string should be "-" if the symbol is to precede the currency value, or "+" if it should appear after the value. Characters within the string can be specified both literally and using " \ " escapes; the following three strings are equivalent:

"+DM"	literal
"+\x44M"	hexadecimal escapes
"+D\115"	octal escapes

All characters following a hash (#) are treated as a comment and ignored up to the end of the line, unless the hash is within a quoted string.

The binary table output is placed in a file named *currency* , within the current directory. This file should be copied or linked to the correct place in the *setlocale* file tree (see *locale* (M)). To prevent accidental corruption of the output data, the file is created with no write permission; if the *montbl* utility is run in a directory containing a write-protected *currency* file, the utility will ask if the existing file should be replaced - any response other than "yes" or "y" will cause **montbl** to terminate without overwriting the existing file.

If the *specfile* argument is missing, the specification information is read from the standard input.

See Also

chrtbl(M), locale(M), msgtbl(M), nl_langinfo(S), numtbl(M), setlocale(S), timtbl(M)

Diagnostics

If the input table file cannot be opened for reading, processing will terminate with the error message, "Cannot open specification file".

Any lines in the specification file which are syntactically incorrect, or contain an unrecognized value instead of CRNCYSTR will cause an error message to be issued to the standard error output, specifying the line number on which the error was detected. The line will be ignored, and processing will continue.

If the output file, *currency*, cannot be opened for writing, processing will terminate with the error message, "Cannot create table file".

Any error conditions encountered will cause the program to exit with a non-zero return code; successful completion is indicated with a zero return code.

Value Added

montbl is an extension of AT&T System V provided in Altos UNIX System V.

mscreen

serial multiscreens utility

Syntax

mscreen [**-s**] [**-n** number] [**-t**]

Description

mscreen allows a serial terminal to have multiple login screens similar to the *multiscreen*(M) console.

Note: For full mscreen support the terminal must have the ability to switch internal screen pages on command and it must retain a separate cursor position for each screen page.

The options are used as follows:

-s Silent mode. This flag suppresses the startup messages, and on ''dumb'' terminals it suppresses the screen switch messages

-n Selects the number of serial multiscreens desired up to the maximum defined for the terminal type.

-t Disables the transparent tty checking. *mscreen* normally exits silently if the terminal device name starts with the characters ''ttyp''. Device names beginning with ''ttyp'' are used as slave devices for *mscreen*. The correct names for the master tty devices begin with ''ptyp''.

mscreen can be used on both ''smart'' and ''dumb'' terminals. Although it is optimized to take advantage of smart terminals with screen memory, *mscreen* also works on dumb terminals, although the screen images are not saved during screen changes. *mscreen* also supports terminals with two (or more) serial ports that are connected to different computers.

mscreen is designed to be invoked from the **.profile** or **.login** files. Use *mscreen* in place of the SHELL variable so that serial multiscreens can be automatic at login time. The ''stop'' and ''quit'' keys allow you to logout from all screens with a single keystroke.

Configuration

mscreen determines the terminal type of the terminal it is invoked from by examining the environment variable TERM. *mscreen* looks in /etc/**mscreencap** or in the filename contained in the environment variable MSCREENCAP to get the capabilities for the terminal type.

The pseudo terminals assigned to the user are automatically determined at startup by *mscreen.* Manual assignment of ttys can be accomplished by creating a file in the user's home directory called *.mscreenrc.*

mscreencap format

mscreencap contains an entry for each terminal type supported. An entry may have several names if the support for several terminal types are the same. Within an entry are the key mappings for each potential pseudo terminal. Each pseudo terminal has a help key string, an input string (the sequence generated by the key that selects this screen), and an optional output string (the sequence to send to the terminal that will cause a page switch). The input and output strings are in a termcap like format: (the backslash and caret are special lead in (escape) characters)

nnn	an octal number, one to three digits are allowed
\n	newline
\r	carriage return
\t	tab
\b	backspace
\f	form feed
\E	escape (hex 1b octal 33).
\	enter backslash as a data character
\^	enter caret as a data character
\^X	ctrl-X where X can be: @ABCDEFGHIJKLMNOPQESTUVWXYZ[]^_ effectively the caret can generate hex 01 through hex 1f.

If a terminal type has no output strings then it is assumed to be a dumb terminal that does not have multiple internal memory pages.

There are five special entries that allow the user to define keys to support the other functions of *mscreen.* They are the help key (which prints a list of all of the keys that are currently available and their functions), the who key (prints the name of the current screen), the stop key (terminates *mscreen* and returns a good (zero) shell return code), and quit key (terminates *mscreen* and returns a bad (non-zero) shell return code and the dummy entry that is used for terminals with multiple ports.

The format is:

```
#this is a comment and may only appear between entries
entryname|alias1|alias1...|aliasn:
    :specialname,helpname,inputstring,pageselectstring:
    :specialname,helpname,inputstring,pageselectstring:
entryname|alias1|alias1...|aliasn:
    :specialname,helpname,inputstring,pageselectstring:
    :specialname,helpname,inputstring,pageselectstring:
```

The specialname is empty for real screen entries. See the provided **/etc/mscreencap** for examples.

.mscreenrc format

.mscreenrc contains a list of ttynames if the user wants to allocate a fixed set of ttys for use:

ttyp0
ttyp1
ttypn

Shell return codes and auto login/logout

mscreen exits with a bad (non-zero) return code if there is an error or when the "quit" key is pressed. The "stop" key causes *mscreen* to exit with a good (zero) return code. This allows users to place *mscreen* in the .login or .profile files. The .login or .profile files should set up an automatic logout if the *mscreen* return code is good (zero). The following is a **csh** sample invocation of *mscreen* for a .login file:

mscreen -n 4
if ($status == 0) logout

The single key logout feature of *mscreen* works as if a normal logout was entered on each pseudo-terminal. A hangup signal is sent to all of the processes on all the pseudo terminals.

Multiple Port Option

mscreen provides a dummy entry type. It allows *mscreen* to be placed in an inactive state while the user uses his terminal to converse through another (physical) io port to another computer. see the provided /etc/*mscreen*termmap for an example. To be used, you must take the example and configure it for your needs.

mscreen Driver

The *mscreen* driver is already installed in the Altos UNIX System V kernel with eight pseudo terminals available for use. You must enable a pseudo terminals to use it. See the link-kit instructions for relinking the kernel to have more available pseudo terminals.

Notes

mscreen has a VTIM timeout of 1/5 second for input strings.

mscreen has a limit of twenty multiscreens per user.

You should not switch screen pages in *mscreen* when output is occurring because if an escape sequence is cut in half it may leave the terminal in an indeterminate state and distort the screen image.

Terminals that save the cursor location for each screen often do not save states such as insert mode, inverse video, and others. For example, you should not change screens if you are in insert mode in vi, and you should not change screens during an inverse video output sequence.

For inactive screens (screens other than the current one) *mscreen* saves the last 2048 characters of data (2K). Data older than this is lost. This limit occasionally results in errors for programs that require a memory of more data than this. The user-defined screen redraw key restores the screen to normal appearance.

mscreen depends on the pseudo terminal device names starting with ttyp for the slave devices and ptyp for the master devices. The number of trailing character in the device name is not significant.

See Also

multiscreen(M), enable(C), ''Adding Ports, Terminals, and Modems'' in the *System Administrator's Guide*

Value Added

mscreen is an extension of AT&T System V provided in Altos UNIX System V.

multiscreen

multiple screens (device files)

Syntax

alt-Fn
alt-ctrl-Fn
alt-shift-Fn
alt-ctrl-shift-Fn

Description

With the *multiscreen* feature, a user can access up to twelve different "screens," each corresponding to a separate device file. Each screen can be viewed one at a time through the **primary monitor** video display.

The number of screens on a system depends upon the amount of memory in the computer. The system displays the number of enabled screens during the boot process.

Access

To see the next consecutive screen, enter:

Ctrl-PrtSc

To move to any screen from any other screen, enter:

alt-F*n* or alt-ctrl-F*n* or alt-shift-F*n*
alt-F*n* or alt-ctrl-F*n* (screens 1-12)
alt-shift-F*n* or alt-ctrl-shift-F*n* (screens 11-16, 7-12)

where *n* is the number of one of the "F" function keys on the primary monitor keyboard. For example:

alt-F2

selects **tty02**, and all output in that device's screen buffer is displayed on the monitor screen.

The second form (using the **SHIFT** key) permits access to screens 11 and 12 on keyboards that have only ten function keys. It is also possible to configure the kernel for up to 16 screens, but 12 is the default.

The function key combinations used to display the various screens are defined in the keyboard mapping file. The **/usr/lib/keyboard/keys** or other *mapkey*(ADM) file can be modified to allow different key combinations to change multiscreens. Use the *mapkey* utility to create a new keyboard map.

Files

/dev/tty[01-12]	multiscreen devices (number available depends on system memory)

See Also

mapkey(ADM), keyboard(HW), screen(HW), serial(HW), stty(C)

Notes

Any system error messages are normally output on the **console** device file (**/dev/console**). When an error message is output, the video display reverts to the **console** device file, and the message is displayed on the screen. The **console** device is the only teletype device open during the system boot sequence and when in single user, or system maintenance mode.

Limitations to the number of multiscreens available on a system does not affect the number of serial lines or devices available. See *serial*(M) for information on available serial devices.

Note that the keystrokes given here are the default, but your keyboard may be different. If so, see *keyboard*(M) for the appropriate substitutes. Also, any key can be programmed to generate the screen switching sequences by using the mapkey utility.

Value Added

multiscreen is an extension of AT&T System V provided in Altos UNIX System V.

numtbl

create a numeric locale table

Syntax

numtbl [table_file]

Description

This utility will create a numeric locale table to be interpreted by the *setlocale* (S) system call.

The *table_file* contains information about the numeric locale in a user readable form.

At present, two pieces of information can be supplied. These are: the character to be used as a decimal place marker (radix character), and the character to be used as a thousands delimiter, for example the commas in 1,000,000. To specify these, there must be lines, in the table file, of the form:

```
DECIMAL=d
THOUSANDS=t
```

Where ''d'' is the character to be used as the decimal place mark and ''t'' is the character to be used as the thousands delimiter. The characters ''d'' and ''t'' may be specified in six different ways. The following lines show different formats for the letter *b*.

98	- decimal
0142	- octal
0x62	- hexadecimal
'b'	- quoted character
'\0142'	- quoted octal
'\x62'	- quoted hexadecimal

Any line starting with a hash (''#'') is treated as a comment.

The output is a file, called *numeric*, which is placed in the current directory. This file is in a form which can be interpreted by the *setlocale(S)* system call. For more information on where this file should be placed, please see *locale(M)*.

If no table file is specified, the information is taken from the standard input. The format of the information is identical.

If either *DECIMAL* or *THOUSANDS* is not specified, its value will default to "." or ",", respectively.

See Also

locale(M), environ(M)

Diagnostics

Any lines of input which are in the wrong format will cause a warning to be issued on the terminal, but will not terminate the program.

"Character syntax error" will be issued on the terminal if the format of the character specification does not match one of those specified above. The program will then terminate.

If the input table file cannot be opened for reading, the program will also terminate with the error message, "Cannot open table file".

If the output file, *numeric,* cannot be opened for writing, the program will terminate with the error message, "Cannot create numeric locale file".

Notes

The thousands delimiter is not currently used within any of the standard Altos UNIX System V libraries or utilities, although it can be accessed by application programs using the *nl_langinfo* (S) function.

The string RADIXCHAR may be used as an alternative to DECIMAL , and THOUSEP as an alternative to THOUSANDS , if required. These alternatives are provided for consistency with the identifiers used by *nl_langinfo* (S).

Value Added

numtbl is an extension of AT&T System V provided in Altos UNIX System V.

powerfail

perform power failure shutdown service

Syntax

/etc/powerfail **[-t** *seconds*]

Description

If the system is equipped with an uninterruptible power supply (UPS), when *init*(M) receives the power failure signal, it invokes */etc/powerfail* (via *inittab*(M)) to shut down various system services during a power failure.

The specific system services to be shut down are specified by shutdown shell scripts in the appropriate **/etc/rc?.d** directory. These shell scripts must have filenames that begin with the letter For example, if the current run level is 2 (multi-user mode), all **/etc/rc2.d/P*** scripts will be executed.

A timeout mechanism is provided in case a particular shut-down script does not exist. In such cases, *powerfail* will abort that script and continue to the next one. The timout value in seconds can be specified on the command line with the **-t** option. If no **-t** option is used, then there is no timeout.

Files

/etc/rc?.d/P*
/etc/inittab
/etc/conf/cf.d/init.base

See Also

init(M), upsconfig(ADM)

Value Added

/etc/powerfail is an extension of AT&T System V provided in Altos UNIX System V.

profile

sets up an environment at login time

Description

The optional file, **.profile**, permits automatic execution of commands whenever a user logs in. The file is generally used to personalize a user's work environment by setting exported environment variables and terminal mode (see *environ*(M)).

When a user logs in, the user's login shell looks for **.profile** in the login directory. If found, the shell executes the commands in the file before beginning the session. The commands in the file must have the same format as if they were entered at the keyboard. Any line beginning with the number sign (#) is considered a comment and is ignored. The following is an example of a typical file:

```
# Tell me when new mail comes in
MAIL=/usr/mail/myname
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
```

Note that the file **/etc/profile** is a system-wide profile that, if it exists, is executed for every user before the user's **.profile** is executed.

Files

$HOME/.profile
/etc/profile

See Also

env(C), login(M), mail(C), sh(C), stty(C), su(C), environ(M)

promain, nopromain

restrict the execution domain of a program

Syntax

auths -r nopromain

Description

The promain feature allows you to control the damage an SUID program can do to your files. An SUID program starts execution with *effective* user ID equal to the owner of the SUID program file and real user ID equal to the invoker of the SUID program. On traditional UNIX systems, an SUID program has complete access to all files, processes, and IPC objects (collectively called resources) to which the invoker *or* the owner has access, because the program can use *setuid*(S) to switch between the invoker and owner user ID. Outside a promain, this power is restricted to resources to which the invoker *and* the owner have access, as described in this section.

The SUID feature is used when one user (or system function) needs to protect files from access except through a well-defined set of programs. An example is the suite of line printer commands, which work with a set of configuration files, status files, and shell scripts to keep track of which print jobs are queued to which printers. Users and line printer administrators use several commands to submit and cancel jobs, change and query the status of printers, and add and remove printers from the system or from active duty. All printer files are owned by the pseudo user **lp** , the user ID which is the owner of all files used by the line printer subsystem, including the printer special devices themselves.

When you invoke the *lp*(C) command to print a file, the program can access the files in the database, but can also access files that you request to be printed because the program can *setuid* to your user ID to access your files. A malicious *lp* program could just as easily look for protected files in your directory hierarchy and copy them to a place protected such that only **lp** could read them. Thus, the fact that you trust *lp* enough to run it as a program means that you trust that it will not abuse the power you give when you run it.

If you run a SUID program on a trusted system and the **promain** kernel authorization is *off*, a promain is created, and the current directory is noted as the *promain root*. Files in the subtree starting at the promain root are said to be *inside* the promain, while files outside that subtree are said to be *outside* the promain. Promains protect a user against a malicious SUID program by restricting the kinds of accesses the program can do outside the promain when running as you (the invoker). When running with the invoker's effective user ID outside the promain, the program can access files if *both* the invoker and the owner have access (public files). Inside the promain, the program has access according to the rules associated with normal UNIX systems.

To run untrustworthy SUID programs with promain protection, do the following:

1. Start a subshell without the **nopromain** authorization, as in this example:

```
$ auths
Kernel authorizations: nopromain,execsuid
$ auths -r nopromain
$ auths
Kernel authorizations: nopromain
```

2. Change directory to one in which you will place all files which you expect the program to modify, and put all the files the program needs there, as in the following example:

```
$ cd tmp
$ cp path/file1 path/file2 .
$ ls -ld . file1 file2
total 10
drwxrwx---   2 drb  mktng     64 Jul 30 07:53 .
-rw-rw----   1 drb  mktng  10345 Jul 30 07:52 file1
-rw-rw----   1 drb  mktng   9200 Jul 30 07:52 file1
$ pwd
/usr/drb/tmp
```

3. Run the program. In **/usr/drb/tmp** , the program has complete access to **file1** , **file2** , and the **/usr/drb/tmp** directory. Assuming that all files owned by **drb** outside the **/usr/drb/tmp** directory are protected, the program has access only if **drb** allows public access.

Note that promain protection is enforced for *all* system actions outside the promain when running under the effective user ID of the invoker. The program cannot read or write files unless they are accessible to the program owner *and* the invoker. The program cannot use *chown*(S) or *chmod*(S) on files outside the promain as **drb** because **drb** and the SUID program owner cannot possibly *both* own the file unless **drb** is the program owner as well. In addition, the program cannot link files outside the promain to a file inside the promain unless the file is accessible to both the invoker and the owner. This protects against a malicious program getting around promain protection by changing the search path used to access the file.

What Promains Guard Against

Promains were designed to guard against one specific type of Trojan

Horse attack, where another user supplies you with a SUID program which steals some of your files into a location or directory to which you do not have access. It also stops the common case of creating a program which is SUID to your user ID by using *setuid*(S) to:

1. Set the program's effective user ID to you (the real user ID),

2. Create a file (which sets you as the owner), and

3. Change the program's mode such that the SUID bit is on.

Promains stop these attacks by only allowing file creation if both the invoker *and* the program owner are allowed an action. Thus, the program will only be able to access your public files (world or group readable) and will never be able to use the SUID attack to create a malicious program (e.g., a shell) which is SUID to you. If your file hierarchy is suitably protected, you can stop attempts to steal data or create damaging files (e.g., programs which have the same name as system programs which you execute because you have included the current directory in your search path).

What Promains Do Not Guard Against

Promains do not protect against running a SUID program which has access to public files or directories in your hierarchies. Neither do they protect against Trojan Horses which are *not* SUID. A program which is not SUID is effectively given access to any objects to which you have access. Thus, running a program which you do not trust is in effect giving your entire discretionary capability to that process. Always suspect a program which is given to you from someone whose motives you have no reason to trust. The newspapers today are full of stories of Trojan Horses and computer viruses (one special case of a Trojan Horse) which were planted by someone who was able to arrange for a user to run a program which did something malicious.

See Also

auths(C), "Maintaining System Security" in the *System Administrator's Guide*

restart

perform power failure recovery service

Syntax

/etc/restart

Description

During a shutsave restart operation (see *upsconfig*(ADM)), *init*(M) invokes */etc/restart* (via *inittab*(F)) to restart various system services.

The specific system services to be restarted are specified by start-up shell scripts in the appropriate **/etc/rc?.d** directory. These shell scripts must have filenames that begin with the letter For example, if the current run level is 2 (multi-user mode), all **/etc/rc2.d/P*** scripts will be executed.

Files

/etc/rc?.d/R*
/etc/inittab
/etc/conf/cf.d/init.base

See Also

init(M), upsconfig(ADM)

Value Added

/etc/restart is an extension of AT&T System V provided in Altos UNIX System V.

rmb

remove extra blank lines from a file

Syntax

/usr/bin/rmb

Description

/usr/bin/rmb acts as a filter to remove any series of blank lines greater than two lines in length. This means that all long sequences of blank lines will be reduced to two blank lines. This is particularly useful for cleaning **nroff** output of blank lines before putting the output in a file.

See Also

man(C), your nroff documentation

Notes

Because */usr/bin/rmb* is a filter, it must be used within a piped command sequence as shown in the following examples:

```
cat infile | /usr/bin/rmb > outfile

nroff infile | /usr/bin/rmb > outfile
```

It cannot be used in the form **/usr/bin/rmb** *filename*.

Also note that Altos UNIX System V is not shipped with *nroff* or other standard UNIX text formatting utilities. These must be purchased separately.

streamio

STREAMS ioctl commands

Syntax

#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;

Description

STREAMS [see *intro*(S)] ioctl commands are a subset of *ioctl*(S) system calls which perform a variety of control functions on *streams*. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream head.* Certain combinations of these arguments may be passed to a module or driver in the *stream.*

fildes is an open file descriptor that refers to a *stream. command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl,* they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the *stream* referenced by *fildes* is linked below a multiplexer, or if *command* is not a valid value for a *stream.*

Also, as described in *ioctl,* STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. This causes subsequent system calls to fail with *errno* set to this value.

Command Functions

The following *ioctl* commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by *arg* onto the top of the current *stream,* just below the *stream head.* It then calls the open routine of the newly-pushed module. On failure, *errno* is set to one of the following values:

[EINVAL] Invalid module name.

[EFAULT] *arg* points outside the allocated address space.

[ENXIO] Open routine of new module failed.

[ENXIO] Hangup received on *fildes*.

I_POP Removes the module just below the *stream head* of the *stream* pointed to by *fildes*. *arg* should be 0 in an I_POP request. On failure, *errno* is set to one of the following values:

[EINVAL] No module present in the *stream*.

[ENXIO] Hangup received on *fildes*.

I_LOOK Retrieves the name of the module just below the *stream head* of the *stream* pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least FMNameSZ+1 bytes long. An [#include <sys/conf.h>] declaration is required. On failure, *errno* is set to one of the following values:

[EFAULT] *arg* points outside the allocated address space.

[EINVAL] No module present in *stream*.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR Flush read queues.

FLUSHW Flush write queues.

FLUSHRW Flush read and write queues.

On failure, *errno* is set to one of the following values:

[ENOSR] Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.

[EINVAL] Invalid *arg* value.

[ENXIO] Hangup received on *fildes*.

I_SETSIG Informs the *stream head* that the user wishes the kernel to issue the SIGPOLL signal [see *signal*(S) and *sigset*(S)] when a particular event has occurred on the *stream* associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.

S_HIPRI A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.

S_OUTPUT The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.

S_MSG A STREAMS signal message that contains the SIGPOLL signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same Stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, *errno* is set to one of the following values:

[EINVAL] *arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.

[EAGAIN] Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, *errno* is set to one of the following values:

[EINVAL] Process not registered to receive the SIGPOLL signal.

[EFAULT] *arg* points outside the allocated address space.

I_FIND Compares the names of all modules currently present in the *stream* to the name pointed to by *arg*, and returns 1 if the named module is present in the *stream*. It returns 0 if the named module is not present. On failure, *errno* is set to one of the following values:

[EFAULT] *arg* points outside the allocated address space.

[EINVAL] *arg* does not contain a valid module name.

I_PEEK Allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf ctlbuf;
struct strbuf databuf;
long          flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures [see *getmsg*(S)] must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to RS_HIPRI, I_PEEK will only look for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the RS_HIPRI flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or RS_HIPRI. On failure, *errno* is set to one of the following values:

[EFAULT] *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

[EBADMSG] Queued message to be read is not valid for I_PEEK

I_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM Byte-stream mode, the default.

RMSGD Message-discard mode.

RMSGN Message-nondiscard mode.

Read modes are described in *read*(S). On failure, *errno* is set to the following value:

[EINVAL] *arg* is not one of the above legal values.

I_GRDOPT Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in *read*(S). On failure, *errno* is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

I_NREAD Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the *ioctl* return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, *errno* is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

I_FDINSERT Creates a message from user specified buffer(s), adds information about another *stream* and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf ctlbuf;
struct strbuf databuf;
long          flags;
int           fildes;
int           offset;
```

The *len* field in the *ctlbuf strbuf* structure [see *putmsg*(S)] must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fildes* in the *strfdinsert* structure specifies the file descriptor of the other *stream*. *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where I_FDINSERT will store a pointer. This pointer will be the address of the read queue structure of the driver for the *stream* corresponding to *fildes* in the *strfdinsert* structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether O_NDELAY has been specified. No partial message is sent. On failure, *errno* is set to one of the following values:

[EAGAIN] A non-priority message was specified, the O_NDELAY flag is set, and the *stream* write queue is full due to internal flow control conditions.

[ENOSR] Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

[EFAULT] *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

[EINVAL] One of the following: *fildes* in the *strfdinsert* structure is not a valid, open *stream* file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*; *offset* does not specify a properly aligned location in the data buffer; an undefined value is stored in *flags*.

[ENXIO] Hangup received on *fildes* of the *ioctl* call or *fildes* in the *strfdinsert* structure.

[ERANGE] The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the *stream head* of the *stream* corresponding to *fildes* in the *strfdinsert* structure. In this case, *errno* will be set to the value in the message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg* and sends that message downstream.

This mechanism is provided to send user *ioctl* requests to downstream modules and drivers. It allows information to be sent with the *ioctl* and will return to the user any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgment message or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one I_STR can be active on a *stream*. Further I_STR calls will block until the active I_STR completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O_NDELAY [see *open*(S)] flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;        /* downstream command */
int     ic_timout;     /* ACK/NAK timeout */
int     ic_len;        /* length of data arg */
char    *ic_dp;        /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver; and *ic_timout* is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I_STR request will wait for acknowledgment before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the *stream* can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream. On failure, *errno* is set to one of the following values:

[ENOSR] Unable to allocate buffers for the *ioctl* message due to insufficient STREAMS memory resources.

[EFAULT] *arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space.

[EINVAL] *ic_len* is less than 0 or *ic_len* is larger than the maximum configured size of the data part of a message or *ic_timout* is less than -1.

[ENXIO] Hangup received on *fildes*.

[ETIME] A downstream *ioctl* timed out before acknowledgment was received.

An I_STR can also fail while waiting for an acknowledgment if a message indicating an error or a hangup is received at the *stream head.* In addition, an error code can be returned in the positive or negative acknowledgment message, in the event the ioctl command sent downstream fails. For these cases, I_STR will fail with *errno* set to the value in the message.

I_SENDFD Requests the *stream* associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a *stream* pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue [see *intro*(S)] of the *stream head* at the other end of the *stream* pipe to which it is connected. On failure, *errno* is set to one of the following values:

[EAGAIN] The sending *stream* is unable to allocate a message block to contain the file pointer.

[EAGAIN] The read queue of the receiving *stream head* is full and cannot accept the message sent by I_SENDFD.

[EBADF] *arg* is not a valid, open file descriptor.

[EINVAL] *fildes* is not connected to a *stream* pipe.

[ENXIO] Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an I_SENDFD *ioctl* over a *stream* pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user id and group id, respectively, of the sending *stream.*

If O_NDELAY is not set [see *open*(S)], I_RECVFD will block until a message is present at the *stream head.* If O_NDELAY is set, I_RECVFD will fail with *errno* set to EAGAIN if no message is present at the *stream head.*

If the message at the *stream head* is a message sent by an I_SENDFD, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, *errno* is set to one of the following values:

[EAGAIN] A message was not present at the *stream head* read queue, and the O_NDELAY flag is set.

[EBADMSG] The message at the *stream head* read queue was not a message containing a passed file descriptor.

[EFAULT] *arg* points outside the allocated address space.

[EMFILE] NOFiles file descriptors are currently open.

[ENXIO] Hangup received on *fildes*.

The following two commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two *streams*, where *fildes* is the file descriptor of the *stream* connected to the multiplexing driver, and *arg* is the file descriptor of the *stream* connected to another driver. The *stream* designated by *arg* gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgment message to the *stream head* regarding the linking operation. This call returns a multiplexer ID number (an identifier used to disconnect the multiplexer, see I_UNLINK) on success, and a -1 on failure. On failure, *errno* is set to one of the following values:

[ENXIO] Hangup received on *fildes*.

[ETIME] Time out before acknowledgment message was received at *stream head.*

[EAGAIN] Temporarily unable to allocate storage to perform the I_LINK.

[ENOSR] Unable to allocate storage to perform the I_LINK due to insufficient STREAMS memory resources.

[EBADF] *arg* is not a valid, open file descriptor.

[EINVAL] *fildes stream* does not support multiplexing.

[EINVAL] *arg* is not a *stream*, or is already linked under a multiplexer.

[EINVAL] The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, I_LINK will fail with *errno* set to the value in the message.

I_UNLINK Disconnects the two *streams* specified by *fildes* and *arg*. *fildes* is the file descriptor of the *stream* connected to the multiplexing driver. *fildes* must correspond to the *stream* on which the *ioctl* I_LINK command was issued to link the *stream* below the multiplexing driver. *arg* is the multiplexer ID number that was returned by the I_LINK. If *arg* is -1, then all Streams which were linked to *fildes* are disconnected. As in I_LINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, *errno* is set to one of the following values:

[ENXIO] Hangup received on *fildes*.

[ETIME] Time out before acknowledgment message was received at *stream head*.

[ENOSR] Unable to allocate storage to perform the I_UNLINK due to insufficient STREAMS memory resources.

[EINVAL] *arg* is an invalid multiplexer ID number or *fildes* is not the *stream* on which the I_LINK that returned *arg* was performed.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, I_UNLINK will fail with *errno* set to the value in the message.

See Also

close(S), fcntl(S), intro(S), ioctl(S), open(S), read(S), getmsg(S), poll(S), putmsg(S), signal(S), sigset(S), write(S)

STREAMS Programmer's Guide
STREAMS Primer

Diagnostics

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

subsystem

security subsystem component description

Description

Altos UNIX System V includes extensions to UNIX that segregate commands and data that are used to implement system services. Many of these commands have been grouped into subsystems. A group of commands and data performing similar security relevant tasks or together protecting a set of resources is termed a *protected subsystem.*

The operating system has the following protected subsystems:

- Memory
- Terminal
- Line Printer
- Backup
- Authentication
- Cron
- Audit

The description of each subsystem includes the following information:

Group and Subsystem Authorization Name
: Each subsystem is associated with a subsystem authorization. The commands and files associated with the subsystem take the subsystem authorization name as their group name. Users wishing to use the subsystem must have the appropriate subsystem authorization.

Commands
: Each subsystem has a set of commands.

Helper Programs
: Some subsystems use helper programs. These are programs which call other programs.

Data Files
: A subsystem's programs use permanent and temporary data files.

The administrative functions associated with each subsystem can be selected from the **sysadmsh** menu. Help information is available with each option.

The Memory Subsystem

The **mem** subsystem authorization is defined to grant users the ability to use the memory subsystem commands to view total system activity. Users without this authorization may only view their own processes. Traditional UNIX allowed any user to view total system activity. This authorization was introduced to allow the administrator to isolate users, and restrict their ability to sense the activity of other users.

Mem Authorization and Group Name

In order to look at information in the **mem** subsystem, an administrator must have the **mem** authorization. The administrator responsible for maintaining users' processes should be the only person with this authorization. This administrator may need to list users' processes in order to select one or more of them for removal (using the *kill*(C) command). The following is a table of command modifications managed by the **mem** authorization:

Command	With Mem	Without mem
ps	lists all processes (standard behavior)	list processes owned by login user ID, or owned by real user ID of current process on current terminal
whodo	lists all processes (standard behavior)	list processes on terminals owned by user
ipcs	lists all objects (standard behavior)	list objects for which user is creator or owner or for which user has read access

sysadmsh Selection

The Memory subsystem does not have a **sysadmsh** selection as the Printer subsystem does. The Memory subsystem includes the system tables that contain information about memory and processes, which is accessed by several commonly-used Altos UNIX System V utilities.

Commands

ps An administrator with **mem** authorization can use the *ps*(C) command to list all users' processes. Using the command without the **mem** authorization shows only those processes associated with the user invoking it.

whodo An administrator with **mem** authorization can use the *whodo*(ADM) command to list processes by terminal. Someone using the command without **mem** authorization sees only the processes associated with his/her terminal.

ipcs An administrator with **mem** authorization can use this command to view active semaphores, shared memory segments and message queues (known collectively as IPC entities). Without **mem** authorization, a user is restricted to viewing IPC entities that they own or created and those which have read permission. Even entities that are writable, but not readable, cannot be displayed.

crash An administrator with **mem** authorization can run the *crash* program to report information on kernel data structures. The report includes security information.

An administrator can search for information by running *crash* and specifying an identifier name.

Helper Programs

timex Because *timex* uses internal kernel data structures, it must be run from an account in the **mem** group.

Accounting Programs

Accounting programs such as *sa*(ADM), *acctcom*(ADM), and *sar*(ADM) also use information in the **mem** subsystem. These programs must be run from an account in the **mem** group.

Data Files

All files through which programs may access kernel memory are protected with owner **root**, group **mem**, and mode -r--r-----. As for all files, the **root** account bypasses the discretionary check on these files, and root programs may violate the System Architecture requirement. All *root* programs (those running with effective ID equal to **root**) must take care when running other programs, because those programs inherit the right to modify the running copy of the TCB. The following files are protected by the **mem** subsystem according to the above owner, group, and mode:

The **terminal** subsystem protects the use of terminals by restricting the use of the *write*(C) and *mesg*(C) commands.

Terminal Authorization and Group Name

In order to send information from one terminal to another, the user sending information must have the **terminal** authorization and the receiving terminal must be configured to accept information from other terminals.

All terminals belong to the **terminal** group. Each terminal is owned by and can only be used by a given user identity.

sysadmsh Selection

The *terminal* subsystem does not control *sysadmsh* functions.

Commands

When an unauthorized user uses the *write* command, any special control codes or escape sequences he sends are trapped and converted to presentable ASCII characters. All control codes are output as

```
^<char>
```

where <char> is the character whose ASCII code is the character sent plus 0100. For instance, ASCII NUL (0), SOH (1), and ACK (6) are output as **^@** (@ is 0100), **^A** (A is 0101) and **^F**, respectively, on the recipient's terminal. The ASCII ESC (033) character writes as ^[and the DEL (0177) character writes as **^?**.

As an example of using the trusted write command, assume there is a hypothetical terminal that silently stores any string between two ASCII DC4 (024) characters. This string is transmitted from the same hypothetical terminal to the computer when the terminal receives a DC2 (022) character. Assume that a devious user knows the recipient of a **write** command has this terminal and tries to corrupt the recipient's session by sending a damaging message. If this user did not have the **terminal** authorization, the recipient would see the message:

```
How are y^Trm *^Tou today^E?
```

The recipient would be alerted to an attempt on his session. In addition, the **terminal** subsystem audits this event so you can locate suspect activity. On the other hand, if the sending user has the **terminal** authorization, the recipient would see the message:

```
How are you today?
```

The following commands are modified to support the terminal subsystem.

Command	With Terminal	Without Terminal
write	unrestricted (standard behavior)	control codes output as ^<char>
mesg	changes sense of group write permission only	same

A person with **terminal** authorization can use the *write*(C) command to write to another terminal and send control codes and escape sequences. A malicious user might use the command to send malicious commands and breach system security.

Without the authorization, a user can use the *write*(C) command, but control codes and escape sequences are displayed on the receiving terminal in their ASCII form, thus warning the recipient of suspicious activity. Such activity is recorded by the audit facilities.

The *mesg y* form of the command allows messages, but sets write permission for the terminal device group that has been set to *terminal* by the login program. The new write command is *SGID to terminal*, which allows it to send characters to user terminals that have used **mesg y**. of the file enough for the *terminal* group to write to the terminal. The new *write* command handles this change. Unlike the less trusted *mesg*, Altos UNIX System V *mesg* never allows any permission to all users.

Data Files

The data files for the *terminal* subsystem are the terminals themselves. They belong to the **terminal** group at the start and end of each session, and all access is denied except to the user. The preferred way for a user to open and close access to a terminal is to use the *mesg* command. When a session is not in progress on a terminal, only the super user can access the device file. Some terminal files are presented below.

/dev/console This is the system console. Use of this terminal as a user terminal is discouraged because:

- Messages from the kernel appear on **/dev/console**. To avoid losing these messages or intermixing them with user messages, it is better to use the console solely for the message output.

- On some systems, physical access to the console is equivalent to having access to the entire system. Use another terminal unless the system configuration prevents this. In any event, allow physical access to **/dev/console** only to the most trusted users of the system.

/dev/tty* Most of the terminals on the system are named **/dev/tty1** , **/dev/tty2** , **/dev/tty3** , ... These devices may at times be owned by a protected subsystem (such as *uucp* or *terminal*) and be unavailable for general use. You have the option of configuring the terminals for login sessions, protected subsystems, or for nothing.

Line Printer Subsystem

The purpose of the *lp* subsystem is to provide an administrative role that has control over printing facilities. Unlike the less trusted version of the **lp** commands, the trusted version does not require a special printer account that owns and executes (with the SUID bit set) all the printer programs. Instead, there is an **lp** group with multiple users as its members.

Authorization/Group Name

The **lp** authorization allows the user to be a printer administrator. This allows multiple Printer administrators. They force the administrator to have a login userid (LUID) of 0 or a login name of **lp** , a scheme that does not allow you much flexibility in account setups or individual accountability.

All printer administrators are allowed to execute some commands that non-authorized users cannot, and can perform certain actions within commands that are restricted from other users. Only administrators may run *accept*, *lpadmin*, *lpmove*, *lpsched*, *lpshut*, *reject* and *topq*. For the other commands, enhancements due to **lp** authorization are detailed under each command heading.

sysadmsh Selection

The *lp* authorization allows access to the printing functions under the **System→Printer** selection as described in the "Using Printers" chapter.

Commands

To determine the invoker, the Printer subsystem command uses the immutable login user ID (LUID). Less trusted versions use various other schemes, all of which could be fooled. The commands listed here perform exactly like their traditional (less trusted) versions except where noted:

accept The *accept* command may only be used by printer administrators.

cancel The less trusted version of cancel allowed any user to cancel any job. The originating user is notified of the cancellation via mail. The trusted version of *cancel* gives this right to printer administrators only. Mail is still sent to the originator when a job is canceled by the printer administrator. Other users can only remove jobs they submitted.

disable The *disable* command operates without change from the less trusted version.

enable The *enable* command operates without change from the less trusted version.

lp The trusted version of the *lp* command, with the **-w** option enabled by you, never writes to the terminal directly as does the less trusted version of *lp*. The trusted version of *lp* knows that the system prohibits direct writing to another user's terminal. Instead, the *write*(C) program is used to send the message; refer to the previous discussion of *write* in the *terminal* subsystem.

The trusted version of the *lp* command creates an output label for each file submitted. The output label contains the system label (the same as seen on most terminals), the owner, group, and mode of the file. To accurately determine the output label, the *lp* command cannot accept input from pipes. This is because the discretionary attributes of a file are not available if the file was accessed on the other end of a pipe. Note that input redirection and temporary files may still be printed.

Printer files are always copied to the printer spool by assuming the **-c** (copy) option, even if the user did not explicitly request it. By doing this, the **lp** subsystem ensures that the file cannot be altered between the time the request was made and the time it is printed. (The less trusted version of *lp* does not guarantee that the file cannot be updated, even while the printer is running.) As added protection, the file being copied is locked during the formation of the output label and the copy operation, so that the file and label output accurately reflects the file being printed.

lpadmin The *lpadmin* command may only be used by printer administrators.

lpforms The *lpforms* command operates without change from the less trusted version.

lpmove The *lpmove* command may only be used by printer administrators.

lpsched The *lpsched* command may only be used by printer administrators. When the *lpsched* command uses a printer device dedicated to the **lp** subsystem, the subsystem guarantees exclusive use of the printer device each time it is used. Any prior activity (outside the **lp** subsystem) on that device is forcibly stopped. In this way, the **lp** subsystem ensures that the file being output is not interspersed with other output, unlike less trusted versions.

lpshut The *lpshut* command may only be used by printer administrators.

lpstat The trusted version of *lpstat* does not display other users' jobs if the invoking user does not have the **lp** authorization. Knowing the jobs of other users is not necessary since unauthorized users cannot hold or cancel those jobs anyway. printer administrators see all printer jobs, and they can hold or cancel any job that has been submitted.

reject The *reject* command may be used only by printer administrators.

topq The *topq* command may be used only by printer administrators.

Data Files

/usr/spool/lp All the files in this file hierarchy have the same formats and purposes as their counterparts in less trusted versions of UNIX. In the trusted version, the files are accessible by any printer administrator, so that the group permissions are the only ones of true importance. In all cases, the spool, its directories, and all data files allow no access to the user population. Hence, a user can be assured that a private file that is spooled for printing cannot be accessed or changed by untrusted users.

Backup Subsystem

The purpose of the *backup* subsystem is to provide a full set of disk and tape management tools without requiring detailed knowledge of

Altos UNIX System V. The backup administrator assumes responsibility of file system maintenance. The backup administrator is responsible for all actions which do not modify the format of file systems, while the **root** account is still responsible for formatting, configuring, and maintaining the consistency of file system disk partitions.

Authorization/Group Name

The user with *backup* authorization, a Backup administrator, may perform file backups. Restorations can only be made by the root user. The following authorizations are defined for the backup subsystem:

Authorization	Type	Purpose
backup	primary	enables system backup command
queryspace	secondary	allows use of *df* program

All disk partitions are protected with owner **root**, group **backup** and mode -r--r-----.. The mount table (**/etc/mnttab**) is publicly readable, modified only by the *mount* command. The *df* program is SGID to backup, enforcing the **queryspace** and **backup** authorizations.

sysadmsh Selection

The *backup* authorization allows access to the backup functions under the *Backups* selection.

Commands

df — The *df* command may only be used by Backup administrators. Otherwise, the options and output format remain the same as the less trusted version.

mkfs — The *mkfs* command may only be used by a member of the *backup* group (or by the super-user, which is discouraged). As always, this command must be used to initialize a filesystem after the partitions are laid out. Immediately after *mkfs* is run, you should run *labelit* to complete the initialization.

labelit — The *labelit* program, documented in *volcopy*(ADM), associates the filesystem with a directory mount point.

Helper Programs

/etc/mount — This program is used by *backupif* to display and modify the mounted file systems.

/etc/fsck — This program is used by *backup* to check and repair filesystems.

/usr/bin/backup — This program is used to copy entire UNIX and XENIX filesystems to either magtape or cartridge tape.

/usr/bin/xbackup — This program is used to copy entire XENIX disk filesystems to either magtape or cartridge tape.

/usr/bin/xrestore — This program is used by replace entire XENIX filesystem images on magtape or cartridge tape to a clean (newly formatted with *mkfs)*

/usr/bin/restore — This program is used by replace entire XENIX or UNIX filesystem images on magtape or cartridge tape.

/usr/bin/cpio — This is the default backup program. **cpio** makes non-filesystem specific copies of filesystem data.

Data Files

/etc/default/filesys
This file contains the relationship between mounted filesystem devices and the directories on which they are mounted (mount points). It is used to display that relationship in both *df* and the *backup* selection. Because altering this file would display erroneous information to backup administrators and reading this file defaults the access protection created for the *backup* subsystem, this file must be accessible to the *backup* group only.

/dev/[r]d[s]k*
These block and character special files are the buffered interfaces to the disk partitions you have set up. They are used for mounting the filesystem they contain onto a directory. The *backup* group must be able to read and write these files. It is a severe security breach if others can access these files in any way.

Authentication Subsystem

The Authentication subsystem provides you with an exhaustive set of account management services. These services are:

- self-checking to prevent dangerous actions, and
- monitored extensively by the auditing system.

Authorization/Group Name

The *auth* authorization allows an Authentication administrator to perform sensitive actions on the Authentication database. This database contains all information on account ownership, types, authorizations, locked status, login times, password change times, and various other parameters.

With the *auth* authorization, an Authentication administrator may alter Authentication parameters for other users. Because this database directly controls the attributes of any account on the system, this subsystem controls user access to your system. The trust you place in the system can be no greater than that placed in the Authentication administrators. Not only must they be trustworthy people, but they must also not leave any uncorrected mistakes when assigning authorizations to the accounts they manage.

sysadmsh Selection

The *auth* authorization allows access to the user account management functions under *Accounts*.

Commands

passwd The *passwd* command in Altos UNIX System V has been greatly enhanced for both security and flexibility. The trusted system checks on system-wide password parameters as well as user-specific ones and, depending on the results found, the user has a choice of choosing their own password or having one chosen for them. You can set each account to do either one of these, or do both. A closely related change is that, regardless of the method for getting the password, you can have the system screen passwords that are probable guesses by intruders. The password selection method, as well as the optional restriction screening, are set by Authentication administrators in *sysadmsh* for a single account or for system-wide use.

login The *login* command is no longer available as a command used in a session to start a new session. Instead, a user must first log out before logging in as another user.

Sublogins are forbidden since the LUID of a session may not change once it is set. This is to guarantee to you that the owner of a session is known at all times. If the *login* program were allowed to be run from a session, the login USERID would have to change and the guarantee would be broken.

The *login* program is still invoked from *getty* to start a user session. The procedure for logging in is almost the same. The user supplies a login name and the system requests a password. Once the password is entered, the system either lets the user log in or rejects the login attempt. A user may be rejected for a number of reasons:

1. The account does not exist.

2. The password was entered incorrectly.

3. The password lifetime has been passed.

4. The number of unsuccessful attempts made to the account has surpassed a system or account threshold.

5. The number of unsuccessful attempts made to the terminal has surpassed a system or terminal threshold.

6. An Authentication administrator has unconditionally locked the account.

Reasons 3 throughh 6 notify the user that the Authentication administrator has locked the account.

If the user enters the correct login name/password combination, the last successful and unsuccessful login times are displayed on the terminal. The user should view the dates and times of each to determine if someone else has used the account. These dates may also be used to determine whether a Trojan horse program is simulating the *login* procedure to obtain a password. A user with doubts about the authenticity of the login dates and times should report it to you. The earlier you take action on this, the better you can use fresh audit trails and people's recollections to find the source of the problem.

su The *su* program has been strengthened a great deal for security. It now uses information from the Authentication database in determining whether or not to allow a user to "switch" to another user. The following rules apply:

- A user cannot use *su* to enter an account that has been locked.

- The *su* command cannot be used as a means to bypass the lock-checking done by *login*, *at*, and *cron*.

newgrp The *newgrp* command operates without change from the less trusted version.

auths The *auths* command is especially tailored for Altos UNIX System V to allow all users to adjust their authorizations. No user can increase authorizations, but one can temporarily decrease authorizations in order to run an untrusted program or to prevent mistakes. More details on the authorizations and syntax are given in the man page for *auths (C).*

Data Files

/usr/adm/sulog

This file keeps track of the history of use of the *su* program. Each line represents an attempt to run the *su* program. The date and time are first recorded on the line. Then, a '-' means the attempt failed; a '+' means the attempt succeeded. After the '-' or '+' code, the terminal of the attempt is provided. Last, the login name (using the login UID) of the invoker of *su* , together with the login name of the (attempted) changed real UID is presented. As an example, the following log excerpt presents some interesting situations:

```
SU 02/29 19:19 + tty?? root-lp
SU 03/01 20:22 + tty2 blf-root
SU 03/04 04:13 + tty2 fred-proj1
SU 03/07 20:30 - tty2 reese-star
SU 03/07 20:30 + tty2 reese-star
SU 03/07 21:38 + modem auth-root
SU 03/07 21:39 + tty2 blf-root
SU 03/07 21:39 - tty7 daa-root
SU 03/07 21:40 - tty7 daa-root
SU 03/07 21:40 - tty7 daa-root
SU 03/07 21:41 - tty7 daa-root
SU 03/07 21:41 - tty7 daa-root
SU 03/07 21:47 + tty2 fred-proj1
```

- Foremost, it appears as though the user **daa** is attempting to break into the *root* account, for there are many unsuccessful attempts (denoted with the '-' attribute) in rapid succession. That should be investigated further.

- The *su* program does not require one to become the *root* user. In the log above, users **root** , *fred* and *reese* chose to assume the identities of other users.

- In the effort by *reese* to become the *star* user, the first attempt failed and the next immediately succeeded. This occurs frequently and is quite natural when users mistype the password of the other account. You should get suspicious, however, when the number of unsuccessful attempts becomes large. Such attempts, like the case with *daa* above, probably means a breach of security.

- The *su* program was used by *root* to enter the *lp* account. This occurrence was detached from any terminal, because of the special terminal designation of *tty??*. This particular case occurred from */etc/rc* where the *lpsched* daemon is run.

The */usr/adm/sulog* file needs attention periodically. It should be examined and then pruned, saving the most recent entries. The entries removed from the file should be archived if possible rather than completely deleted.

/tcb/files/auth
This directory consists of subdirectories that contain private account data for all the accounts in the system. There is a file for each account. Because of the sensitive nature of the data here, all these files are completely protected from the users.

/etc/auth/system
This directory contains the system-wide authorization data for the machine. The */etc/auth/system* directory contains the Terminal Control database, the File Control database, the Command Control database and the System Defaults database. This information is accessible to the users but not writable. The */etc/auth/subsystems* directory contains one file per protected subsystem, each containing the user permissions for that protected subsystem. This permissions file may only be read by the programs that are part of that protected subsystem, and is written by the auth user.

cron Subsystem

The purpose of the *cron* subsystem is to allow *cron*, *at*, and *batch* services that are audited as closely as normal login sessions. The cron subsystem provides a useful interface for controlling these facilities.

Authorization/Group Name

The authorization for the *cron* subsystem is given to cron administrators that are allowed to view or alter the authority for users to run the services associated with the *cron* subsystem. A user may run the programs of the *cron* subsystem (excluding the use of the *sysadmsh* selections) without the authorization, so long as a cron administrator has granted the authority.

sysadmsh Selection

The *cron* authorization allows access to the process management functions under **Jobs** .

Commands

at, batch, crontab
: These *at* commands operate without change from the less trusted version, except that the LUID (login UID), rather than the real UID, is used by *at* in determining the user. Because the LUID cannot be altered during a session, it promotes better accountability. *at* and *batch* jobs run with all of the login, real, and effective UIDs set to that of the login user.

Helper Programs

/tcb/lib/cron
: This is the *cron* daemon that actually runs all *at*, *batch*, and *crontab* jobs. The *at*, *batch*, and *crontab* commands merely queue the jobs for the *cron* daemon to run. This daemon validates the account (ensures the account is not locked) before running the job.

Data Files

Although enumerated here, these data files are not manipulated directly by the *cron* administrator because of the arcane rules historically applied to them by the *cron* subsystem programs. Instead, the *sysadmsh* provides a more coherent interface, reducing the possibility that users or permissions are set up incorrectly.

/usr/lib/cron
: This is the directory containing all the *cron* administrative files.

/usr/lib/cron/at.allow
: This file lists the users allowed to execute the *at* or *batch* programs. If this file exists, it is used to determine the user's authority.

/usr/lib/cron/at.deny
: This file lists the users denied access

to the *at* or *batch* programs. If **/usr/lib/cron/at.allow** does not exist, */usr/lib/cron/at.deny* is used to determine the user's authority. You should be aware that an empty at.deny file permits access for all users.

/usr/lib/cron/cron.allow This file lists the users allowed to execute the *crontab* program. If this file exists, it is used to determine the user's authority.

/usr/lib/cron/cron.deny This file lists the users denied access to the *crontab* program. If */usr/lib/cron/cron.allow* does not exist, */usr/lib/cron/cron.deny* is used to determine the user's authority. You should be aware that an empty cron.deny file permits access for all users.

/usr/lib/cron/.proto This file contains a list of commands that are executed before every *at* job. It contains commands primarily used to fix and restrict the environment of the user before running the job submitted.

/usr/lib/cron/.proto.b This file contains a list of commands that are executed before every *batch* job. It contains commands primarily used to fix and restrict the environment of the user before running the job submitted.

/usr/lib/cron/log This is a log of all *at*, *batch*, and *crontab* activity reported by the *cron* daemon since the system was rebooted. It provides an accurate ASCII log of all user initiated non-terminal activity. If the system is up for a very long time and there are many job submissions or crontab activity, this file should be periodically examined, pruned, and archived.

/usr/lib/cron/OLDlog This is the log associated with the last time the system was up. Upon startup, the *cron* daemon moves any **/usr/lib/cron/log** file here.

/usr/spool/cron This is the root of the subtree where all *at*, *crontab*, and *batch* jobs are stored. *at* and *batch* jobs are automatically cleared when they have finished executing. The *-r* option of *crontab* removes a *crontab* job.

Audit Subsystem

The purpose of the *audit* subsystem is to provide an administrative role that has control over auditing facilities.

Authorization/Group Name

The **audit** authorization allows the user to be the audit administrator. The audit adminsitrator can enable and disable auditing, examine audit records, generate reports and alter audit parameters.

sysadmsh Selection

The *audit* authorization allows access to the audit functions under the **System→Audit** selection as described in the "Maintaining System Security" chapter.

Commands

auditcmd The command interface for audit subsystem activation, termination, statistic retrievel, and subsystem notification.

auditd The *auditd* utility is the daemon that runs when auditing is enabled.

reduce This program performs audit data analysis and rduction.

Data Files

/tcb/files/audit/audit_parms
Audit parameters file.

/tcb/files/audit/*
Audit log file directory.

/tcb/audittmp Audit compaction file directory.

See Also

auditcmd(ADM), auditd(ADM), authck(ADM), integrity(ADM), reduce(ADM), chg_audit(ADM), auths(C), authcap(F), audit(HW)
"Maintaining System Security" in the *System Administrator's Guide*

Value Added

subsystem is an extension of AT&T System V provided in Altos UNIX System V.

sxt

pseudo-device driver

Description

sxt is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the *sxt* driver. *sxt* acts as a discipline manipulating a *real tty* structure declared by a real device driver. The *sxt* driver is currently only used by the *shl*(C) command.

Virtual ttys are named **/dev/sxt???** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing.

Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(S) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(M), with the addition of the following:

TIOCEXCL Set *exclusive use* mode: no further opens are permitted until the file has been closed.

TIOCNXCL Reset *exclusive use* mode: further opens are once again permitted.

The second group are directives to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

EINVAL The argument is out of range.

ENOTTY The command was not issued from a real tty.

ENXIO *linesw* is not configured with *sxt*.

EBUSY An SXTIOCLINK command has already been issued for this real *tty*.

ENOMEM There is no system memory available for allocating the virtual tty structures.

EBADF Channel 0 was not opened before this call.

SXTIOCSWTCH Set the controlling channel. Possible errors include:

EINVAL An invalid channel number was given.

EPERM The command was not executed from channel 0.

SXTIOCWF Cause a channel to wait until it is the controlling channel. This command will return the error, *EINVAL*, if an invalid channel number is given.

SXTIOCUBLK Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.

SXTIOCSTAT Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.

SXTIOCTRACE Enable tracing. Tracing information is written to the console. This command has no effect if tracing is not configured.

SXTIOCNOTRACE Disable tracing. This command has no effect if tracing is not configured.

Files

/dev/sxt??[0-7]	virtual tty devices
/usr/include/sys/sxt.h	driver specific definitions

See Also

shl(C), stty(C), ioctl(S), open(S), termio(M)

systty

system maintenance device

Description

The file **/dev/systty** is the device on which system error messages are displayed. The actual physical device accessed via **/dev/systty** is selected during boot, and is typically the device used to control the bootup procedure. The default physical device **/dev/systty** is determined by *boot*(HW) when the system is brought up.

Initially **/dev/console** is linked to **/dev/systty**.

Files

/dev/systty

See Also

boot(HW), console(M)

term

conventional names for terminals

Description

These names are used by certain commands [e.g., *man*(C), *tabs*(C), *tput*(C), *vi*(C) and *curses*(S)] and are maintained as part of the shell environment in the environment variable **TERM** [see *sh*(C), *profile*(F), and *environ*(M)].

Entries in *terminfo*(F) source files consist of a number of comma-separated fields. [To obtain the source description for a terminal, use the **-I** option of *infocmp*(ADM).] White space after each comma is ignored. The first line of each terminal description in the *terminfo*(F) data base gives the names by which *terminfo*(F) knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal [this is the one to use to set the environment variable **TERMINFO** in *$HOME/.profile*; see *profile*(F)], the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen. For example, for the AT&T 4425 terminal, the root name is **att4425**. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from [a-z0-9], make up a basic terminal name. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, an AT&T 4425 terminal in 132 column mode would be **att4425-w**. The following suffixes should be used where possible:

Suffix	**Meaning**	**Example**
-w	Wide mode (more than 80 columns)	att4425-w
-am	With auto. margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-n	Number of lines on the screen	aaa-60

-na	No arrow keys (leave them in local)	c100-na
-np	Number of pages of memory	c100-4p
-rv	Reverse video	att4415-rv

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (e.g., **-w**), it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the *terminfo*(F) data base unique. Terminal entries that are present only for inclusion in other entries via the **use=** facilities should have a '+' in their name, as in **4415+nl**.

Some of the known terminal names may include the following (for a complete list, type: **ls -C /usr/lib/terminfo/?**):

2621,hp2621	Hewlett-Packard 2621 series
2631	Hewlett-Packard 2631 line printer
2631-c	Hewlett-Packard 2631 line printer - compressed mode
2631-e	Hewlett-Packard 2631 line printer - expanded mode
2640,hp2640	Hewlett-Packard 2640 series
2645,hp2645	Hewlett-Packard 2645 series
3270	IBM Model 3270
33,tty33	AT&T TELETYPE Model 33 KSR
35,tty35	AT&T TELETYPE Model 35 KSR
37,tty37	AT&T TELETYPE Model 37 KSR
4000a	Trendata 4000a
4014,tek4014	TEKTRONIX 4014
40,tty40	AT&T TELETYPE Dataspeed 40/2
43,tty43	AT&T TELETYPE Model 43 KSR
4410,5410	AT&T 4410/5410 terminal in 80-column mode - version 2
4410-nfk,5410-nfk	AT&T 4410/5410 without function keys - version 1
4410-nsl,5410-nsl	AT&T 4410/5410 without pln defined
4410-w,5410-w	AT&T 4410/5410 in 132-column mode
4410v1,5410v1	AT&T 4410/5410 terminal in 80-column mode - version 1
4410v1-w,5410v1-w	AT&T 4410/5410 terminal in 132-column mode - version 1
4415,5420	AT&T 4415/5420 in 80-column mode
4415-nl,5420-nl	AT&T 4415/5420 without changing labels
4415-rv,5420-rv	AT&T 4415/5420 80 columns in reverse video
4415-rv-nl,5420-rv-nl	AT&T 4415/5420 reverse video without changing labels
4415-w,5420-w	AT&T 4415/5420 in 132-column mode
4415-w-nl,5420-w-nl	AT&T 4415/5420 in 132-column mode without changing labels
4415-w-rv,5420-w-rv	AT&T 4415/5420 132 columns in reverse video
4415-w-rv-nl,5420-w-rv-nl	AT&T 4415/5420 132 columns reverse video without changing labels
4418,5418	AT&T 5418 in 80-column mode
4418-w,5418-w	AT&T 5418 in 132-column mode
4420	AT&T TELETYPE Model 4420
4424	AT&T TELETYPE Model 4424
4424-2	AT&T TELETYPE Model 4424 in display function group ii
4425,5425	AT&T 4425/5425
4425-fk,5425-fk	AT&T 4425/5425 without function keys
4425-nl,5425-nl	AT&T 4425/5425 without changing labels in 80-column mode
4425-w,5425-w	AT&T 4425/5425 in 132-column mode
4425-w-fk,5425-w-fk	AT&T 4425/5425 without function keys in 132-column mode
4425-nl-w,5425-nl-w	AT&T 4425/5425 without changing labels in 132-column mode
4426	AT&T TELETYPE Model 4426S

450	DASI 450 (same as Diablo 1620)
450-12	DASI 450 in 12-pitch mode
500,att500	AT&T-IS 500 terminal
510,510a	AT&T 510/510a in 80-column mode
513bct,att513	AT&T 513 bct terminal
5320	AT&T 5320 hardcopy terminal
5420_2	AT&T 5420 model 2 in 80-column mode
5420_2-w	AT&T 5420 model 2 in 132-column mode
5620,dmd	AT&T 5620 terminal 88 columns
5620-24,dmd-24	AT&T TELETYPE Model DMD 5620 in a 24x80 layer
5620-34,dmd-34	AT&T TELETYPE Model DMD 5620 in a 34x80 layer
610,610bct	AT&T 610 bct terminal in 80-column mode
610-w,610bct-w	AT&T 610 bct terminal in 132-column mode
7300,pc7300,unix_pc	AT&T UNIX PC Model 7300
735,ti	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
pt505	AT&T Personal Terminal 505 (22 lines)
pt505-24	AT&T Personal Terminal 505 (24-line mode)
sync	generic name for synchronous TELETYPE Model 4540-compatible terminals

Commands whose behavior depends on the type of terminal should accept arguments of the form **-T***term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **TERM** , which, in turn, should contain *term*.

Files

/usr/lib/terminfo/? compiled terminal description data base

See Also

curses(S), profile(F), terminfo(M), terminfo(F), environ(M), infocmp(ADM), sh(C), stty(C), tabs(C), tput(C), tplot(ADM), vi(C)

Notes

Not all programs follow the above naming conventions.

terminals

list of supported terminals

Description

The following list, derived from the file **/etc/termcap**, shows the terminal name (suitable for use as a TERM shell variable), and a short description of the terminal. The advice in *termcap*(F) will assist users in creating termcap entries for terminals not currently supported.

Name	Terminal
1200	terminet 1200
1620	diablo 1620
1640	diablo 1640
2392	239x series
2392an	hp 239x in ansi mode
2392ne	239x series
2621	hp 2621
2621k45	hp 2621 with 45 keyboard
2621nl	hp 2621 with no labels
2621nt	hp 2621 w/no tabs
2621wl	hp 2621 with labels
2622	hp 2622
262x	hp 262x series
2640	hp 2640a
2640b	hp 264x series
300	terminet 300
3045	datamedia 3045a
33	model 33 teletype
37	model 37 teletype
40	teletype dataspeed 40/2
4025	tektronix 4024/4025/4027
4025-17	tek 4025 17 line window
4025-17ws	tek 4025 17 line window in workspace
4025ex	tek 4025 w/!
43	model 43 teletype
515	AT&T-IS 515 terminal in native mode
5410	5410 terminal 80 columns
5410-nfk	version 1 tty5410 entry without function keys
5410132	5410 132 columns
5420132	5420 132columns
5425	AT&T Teletype 5425 80 columns
5425-w	AT&T Teletype 5425 132 columns

610bct	AT&T 610; 80 column; 98key keyboard
615mt	AT&T 615; 80 column; 98key keyboard
620mtg	AT&T 620; 80 column; 98key keyboard
7900	NCR 7900-1
8001	intecolor
912b	new televideo
925	newer televideo
925so	newer televideo with attribute byte workaround
ATT5620	5620 terminal 88 columns
Ma2	Ampex Model 232 / 132 lines
TWO	Altos Computer Systems II
a980	adds consul 980
aa	ann arbor
aaa	ann arbor ambassador/48 lines
aaa30	ann arbor ambassador 30/destructive backspace
aaa48db	ann arbor ambassador 48/destructive backspace
aaadb	ann arbor ambassador 48/destructive backspace
act5s	skinny act5
adds	adds viewpoint
adds25	adds regent 25 with local printing
adm11	lsi adm11
adm12	lsi adm12
adm2	lsi adm2
adm3	lsi adm3
adm31	Lear Siegler ADM31
adm3a	lsi adm3a
adm3a+	lsi adm3a+
adm3a19.2	lsi adm3a at 19.2 baud
adm3aso	lsi adm3a with { } for standout
adm42	lsi adm42
adm5	lsi adm5
aj830	anderson jacobson
altos3	Altos III
altos4	Altos IV
altos5	Altos V
am219w	Ampex 132 Cols
amp219	Ampex with Automargins
amp232	Ampex Model 232
ampex	ampex dialogue 80
ansi	Ansi standard crt
ansi-nam	Ansi standard crt without automargin
arpanet	network
at386	at/386 console
at386-m	at/386 console
atarist	Atari ST vt52
att513	AT&T-IS 513 Business Communications Terminal 80 columns
att513-w	AT&T-IS 513 Business Communications Terminal 132 columns

att605	AT&T 605 BCT
att630	AT&T 630 windowing terminal
bct500	teletype 5541
bh3m	beehiveIIIm
big2621	48 line 2621
c100	concept 100
c1004p	c100 w/4 pages
c100rv	c100 rev video
c100rv4p	c100 w/4 pages
c100rv4pna	c100 with no arrows
c100rv4ppp	c100 with printer port
c100rvs	slow reverse concept 100
c100s	slow concept 100
c3102	cromemco 3102
carlock	klc
cci	cci 4574
cdc456	cdc
cdc456tst	cdc456tst
cdi	cdi1203
cie467	C.Itoh 467, 414 Graphics
cit80	c.itoh 80
cit80nam	C.Itoh 80 without automargins
compucolor	compucolorII
d132	datagraphix 132a
datapoint	datapoint 3360
delta	delta data 5000
dg	data general 6053
digilog	digilog 333
dm1520	datamedia 1520
dm1521	datamedia 1521
dm2500	datamedia 2500
dm3025	datamedia 3025a
dmterm	Tandy deskmate terminal
dosansi	ANSI.SYS standard crt
dt100	Tandy DT-100 terminal
dt100w	Tandy DT-100 terminal
dt200	Tandy DT-200
dt80	datamedia dt80/1
dt80132	datamedia dt80/1 in 132 char mode
dtc300s	dtc 300s
du	dialup
dumb	unknown
dw1	decwriter I
dw2	decwriter II
ep40	execuport 4000
ep48	execuport 4080
esp925	esprit tvi925 emulation
espHA	esprit 6310 in hazeltine emulation mode

ethernet	network
exidy	exidy sorcerer as dm2500
fos	Fortune system
fox	perkin elmer 1100
free100	liberty freedom 100
free110	Freedom 110
ft1024	Forward Technology graphics controller
gt40	dec gt40
gt42	dec gt42
h1500	hazeltine 1500
h1510	hazeltine 1510
h1520	hazeltine 1520
h1552	hazeltine 1552
h1552rv	hazeltine 1552 reverse video
h19	heathkit h19 w/ function keypad
h19a	heathkit h19 ansi mode
h19nk	heathkit w/numeric keypad (not function keys)
h2000	hazeltine 2000
hp	hp 264x series
hp2626	hp 2626
hp2648	HP 2648a graphics terminal
hpansi	Hewlett Packard 700/44 in HP-PCterm mode, PC character set
hpansi-24	HP 700/44 in HP-PCterm 24 line mode, PC character set
hpex	hp extended capabilites
hpsub	hp terminals -- capability subset
i100	General Terminal 100A (formerly Infoton 100)
ibm3101	IBM 3101-10
ibm3151	3151
ibm3161	3161
ibm3163	3163
ibm3164	3164
ibm5151	ibm console
ibmcons	Ansi standard with EGA
ibmcons-43	Ansi EGA console in 43 line mode
intext	ISC modified owl 1200
ipc	Intel IPC
k10	Kaypro 10
kn	kt70pcix
kt7ix	kimtron kt-7
lisa	apple lisa xenix console display (white on black)
m100	radio shack model 100
macterm	macintosh MacTerm in vt-100 mode
macterm-nam	MacTerm in vt-100 mode with automargin NOT set
mdl110	cybernex mdl-110
microb	micro bee series
microterm	microterm act iv
microterm5	microterm act v

mime	microterm mime1
mime2a	microterm mime2a (emulating an enhanced vt52)
mime2as	microterm mime2a (emulating an enhanced soroc iq120)
mime3a	mime1 emulating 3a
mime3ax	mime1 emulating enhanced 3a
mimefb	full bright mime1
mimehb	half bright mime1
mt70	morrow mt70
nabu	nabu terminal
netx	netronics
nucterm	NUC homebrew
oadm31	old adm31
omron	Omron 8025AG
ot80	onyx ot80
owl	perkin elmer 1200
pe550	perkin elmer 550
pixel	Pixel terminal
plasma	plasma panel
pt1500	Convergent Technologies PT
pt210	Tandy TRS-80 PT-210 printing terminal
qume5	Qume Sprint 5
qvt101	Qume QVT-101 vers c
qvt101+	Qume QVT-101 Plus vers c
qvt101+so	Qume QVT-101+ with protected mode/standout
qvt101b	QVT-101 with cursor set to blinking underline
qvt102	Qume QVT 102
qvt103	Qume QVT-103
qvt108	QVT-108
qvt109	QVT-109
qvt119	Qume QVT-119
qvt119+	Qume QVT-119 Plus vers c
qvt201	Qume QVT-201
qvt202	Qume QVT-202
qvt203	Qume QVT 203 PLUS
regent	adds regent series
regent100	adds regent 100
regent20	adds regent 20
regent25	adds regent 25
regent25a	adds regent 25a
regent40	adds regent 40
regent60	adds Regent 60
regent60na	regent 60 w/no arrow keys
rx303	rexon 303 terminal
sb1	beehive super bee
sb2	fixed superbee
sexidy	exidy smart
sk8620	Seiko 8620
soroc	Soroc 120

sun	Sun Microsystems Workstation console
sun-cmd	Sun Microsystems Workstation console with scrollable history
sun-nic	Sun Microsystems Workstation console without insert character
sun1	old Sun Microsystems Workstation console
superbeeic	super bee with insert char
svt100	1220/PC, Sperry in VT100 mode
svt1210	Sperry 1210, standard setup
svt1220	Sperry 1220, standard setup
svt52	1210/1220/PC, Sperry in VT52 mode
switch	intelligent switch
swtp	southwest technical products ct82
t1061	teleray 1061
t1061f	teleray 1061 with fast PROMs
t3700	dumb teleray 3700
t3800	teleray 3800 series
td200	Tandy 200
tek	tektronix 4012
tek4013	tektronix 4013
tek4014	tektronix 4014
tek4014sm	tektronix 4014 in small font
tek4015	tektronix 4015
tek4015sm	tektronix 4015 in small font
tek4023	tektronix 4023
tek4107	tektronix 4107
teletec	Teletec Datascreen
terak	Terak emulating Datamedia 1520
ti	ti silent 700
ti745	ti silent 745
ti924	Texas Instruments 924 VDT 7 bit
ti924-8	Texas Instruments 924 VDT 8 bit
ti926	Texas Instruments 926 VDT
ti931	Texas Instruments 931 VDT
trs100	Tandy TRS-80 Model 100
trs16	Tandy trs-80 model 16 console
trs600	Tandy Model 600
tty4420	teletype 4420
tty4424	teletype 4424
tty4424-w	teletype 4424 in display function group ii
tty5410	Teletype 5410 terminal in 80 column mode
tty5410-w	Teletype 5410 in 132 column mode
tvi910	old televideo 910
tvi910+	televideo 910 PLUS
tvi912	old televideo
tvi9220	Televideo 9220 w/status line @ bottom
tvi9220w	Televideo 9220 132 col w/status line @ bottom
tvi924	televideo924

tvi950	televideo950
tvi950-2p	tvi 950 w/2 pages
tvi950-4p	tvi 950 w/4 pages
tvi950-ap	tvi 950 w/alt pages
tvi950b	bare tvi950 no is
tvi950ns	tvi950 w/no standout
v50	Visual 50 emulation of DEC VT52
v55	Visual 55 emulation of DEC VT52 (called V55)
vi200	visual 200 with function keys
vi200f	visual 200 no function keys
vi200ic	visual 200 using insert char
vi200rv	visual 200 reverse video
vi200rvic	visual 200 reverse video using insert char
vi50	Visual 50 in ADDS viewpoint emulation
vi55	Visual 55 using ADDS emulation
vis613	Visual 613
vs100	xterm terminal emulator
vs100s	xterm terminal emulator (small screen 24x80)
vt100	dec vt100
vt100n	vt100 w/no init
vt100nam	DEC VT100 without automargins
vt100s	dec vt100 132 cols 14 lines
vt100w	dec vt100 132 cols
vt102	dec vt102
vt131	dec vt131
vt132	vt-132
vt220	dec vt220 generic
vt220d	DEC VT220 in vt100 mode with DEC function key labeling
vt50	dec vt50
vt50h	dec vt50h
vt52	dec vt52
vt52so	dec vt52 with brackets added for standout use
vtz	Zilog vtz 2/10
w2110A	Wang 2110 Asynch Data Entry Terminal - 80 column
ws584	Olivetti WS584
ws584fr	Olivetti WS584 with French keyboard
ws584gr	Olivetti WS584 with German keyboard
ws584nr	Olivetti WS584 with Norwegian/Danish keyboard
ws584sp	Olivetti WS584 with Spanish keyboard
ws584sw	Olivetti WS584 with Swedish/Finnish keyboard
ws584uk	Olivetti WS584 with U.K. keyboard
ws584us	Olivetti WS584 with U.S.A. keyboard
ws685	Olivetti WS685
wy100	wyse 100
wy120	Wyse 120
wy120-25	Wyse 120 80-column 25-lines
wy120-vb	Wyse 120 Visible bell

wy120-wvb	wyse120-wvb
wy120w	Wyse 120 132-column
wy120w-25	Wyse 120 132-column 25-lines
wy150	Wyse 150
wy150-25	Wyse 150 80-column 25-lines
wy150-vb	Wyse 150 Visible bell
wy150-wvb	wyse150-wvb
wy150w	Wyse 150 132-column
wy150w-25	Wyse 150 132-column 25-lines
wy30	Wyse WY-30 in wy30 mode
wy30-vb	wyse 30 Visible bell
wy350	Wyse 350 80 column color terminal emulating wy50
wy350-vb	wyse 350 Visible bell
wy350-wvb	wyse 350 132-column Visible bell
wy350w	Wyse 350 132 column color terminal emulating wy50
wy50	Wyse 50/80 Wyse WY-50 with 80 column screen
wy50-wvb	wyse 50 132-column Visible bell
wy50l	Wyse WY-60 with 80 column/43 line screen in WY50+ mode
wy50n	Wyse WY-50 - 80 column screen, no automargin
wy50vb	Wyse WY-50/80vb Wyse WY-50/80 with visible bell
wy50w	Wyse WY-50/132 Wyse WY-50 with 132 column screen
wy60	Wyse WY-60 with 80 column/24 line screen in wy60 mode
wy60-25	wyse 60 80-column 25-lines
wy60-42	wyse 60 80-column 42-lines
wy60-43	wyse 60 80-column 43-lines
wy60-vb	Wyse 60 Visible bell
wy60ak	Wyse 60 in wy60 mode with ANSI arrow keys +
wy60w	Wyse WY-60 with 132 column/24 line screen in wy60 mode
wy60w-25	wyse 60 132-column 25-lines
wy60w-42	wyse 60 132-column 42-lines
wy60w-43	wyse 60 132-column 43-lines
wy60w-vb	Wyse 60 132-column Visible bell
wy75	Wyse WY-75 with 80 column line
wy75-mc	wyse 75 with magic cookies
wy75-vb	wyse 75 with visible bell
wy75-wvb	wyse 75 with visible bell 132 columns
wy75ap	Wyse WY-75 with Applications and Cursor keypad modes
wy75w	Wyse WY-75 in 132 column mode
wy75x	Wyse WY-75 with 132 column lines in vi editor mode
wy85	Wyse 85 in 80 column mode, vt100 emulation
wy85-vb	wyse 85 with visible bell
wy85-wvb	wyse 85 with visible bell 132-columns

wy85w	Wyse 85 in 132 column mode, vt100 emulation
wy85w	wyse 85 in 132-column mode
wy99gt	Wyse 99gt
wy99gt-25	wyse 99gt 80-column 25-lines
wy99gt-25-w	wyse 99gt 132-column 25-lines
wy99gt-vb	Wyse 99gt Visible bell
wy99gt-w	wyse 99gt 132-column
wy99gt-w-vb	wyse99gt-wvb
wyse120ak	Wyse 120 with ANSI key values
x1720	xerox 1720
xitex	xitex sct-100
z29	zenith z29
z39	Zenith Z-39
zen30	zentec 30
zen40	zentec 40
zen50	zentec 50
zephyr	zentec zephyr220 in vt100 mode
zephyrnam	zentec zephyr220 in vt100 mode w/out automargins

Files

/etc/termcap

See Also

tset(C), environ(M), termcap(F)

terminfo

terminal capability data base

Syntax

/usr/lib/terminfo/? /*

Description

terminfo is a compiled database [see *tic*(C)] describing the capabilities of terminals. Terminals are described in *terminfo* source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used, for example, by *vi*(C) and *curses*(S), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the **-I** option of *infocmp*(ADM). When doing an *infocmp* for the terminal you are on, there is no difference between *infocmp* and *infocmp -I.*

Entries in *terminfo* source files consist of a number of fields separated by commas. White space after each comma is ignored. The first line of each terminal description in the *terminfo* database gives the name by which *terminfo* knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal [this is the one to use to set the environment variable **TERM** in *$HOME/.profile*; see *profile*(F)]; the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. See *term*(M) for examples and more information on choosing names and synonyms.

PART 1: TERMINAL CAPABILITIES

Capabilities in *terminfo* are of three types: boolean capabilities (which show that the terminal has some particular feature), numeric capabilities (which specify the size of the terminal or particular features), and string capabilities (which provide a sequence that can be used to perform particular terminal operations).

In the following tables, a **Variable** is the name by which a C programmer accesses a capability (at the *terminfo* level). A **Capname** is the short name for a capability used in the source description. It is used by a person updating the database and by the *tput*(C) command when asking what the value of the capability is for a particular terminal. A **Termcap Code** is a two-letter code that corresponds to the old *termcap* capability name.

Capability names have no hard length limit, but an informal limit of five characters has been adopted to keep them short. Whenever possible, names are chosen to be the same as or similar to those specified by the ANSI X3.64-1979 standard. Semantics are also intended to match those of the ANSI standard.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the following table, have names beginning with **key_**. The following indicators may appear at the end of the **Description** for a variable.

(G)
: indicates that the string is passed through **tparm( )** with parameters (parms) as given ($\#_i$)

(*)
: indicates that padding may be based on the number of lines affected

($\#_i$)
: indicates the i^{th} parameter

Booleans

Variable	Cap-name	Termcap Code	Description
auto_left_margin	bw	bw	**cub1** wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
back_color_erase	bce	be	Screen erased with background color
can_change	ccc	cc	Terminal can re-define existing color
ceol_standout_glitch	xhp	xs	Standout not erased by overwriting (hp)
col_addr_glitch	xhpa	YA	Only positive motion for **hpa/mhpa** caps
cpi_changes_res	cpix	YF	Changing character pitch changes resolution
cr_cancels_micro_mode	crxm	YB	Using **cr** turns off micro mode
eat_newline_glitch	xenl	xn	Newline ignored after 80 columns (*Concept*)
erase_overstrike	eo	eo	Can erase overstrikes with a blank
generic_type	gn	gn	Generic line type (e.g., dialup, switch)
hard_copy	hc	hc	Hardcopy terminal
hard_cursor	chts	HC	Cursor is hard to see
has_meta_key	km	km	Has a meta key (shift, sets parity bit)
has_print_wheel	daisy	YC	Printer needs operator to change character set
has_status_line	hs	hs	Has extra "status line"
hue_lightness_saturation	hls	hl	Terminal uses only HLS color notation (Tektronix)
insert_null_glitch	in	in	Insert mode distinguishes nulls
lpi_changes_res	lpix	YG	Changing line pitch changes resolution
memory_above	da	da	Display may be retained above the screen
memory_below	db	db	Display may be retained below the screen
move_insert_mode	mir	mi	Safe to move while in insert mode
move_standout_mode	msgr	ms	Safe to move in standout modes
needs_xon_xoff	nxon	nx	Padding won't work, xon/xoff required
no_esc_ctlc	xsb	xb	Beehive (f1=escape, f2=ctrl C)
no_pad_char	npc	NP	Pad character doesn't exist
non_dest_scroll_region	ndscr	ND	Scrolling region is non-destructive
non_rev_rmcup	nrrmc	NR	**smcup** does not reverse **rmcup**
over_strike	os	os	Terminal overstrikes on hard-copy terminal
prtr_silent	mc5i		
row_addr_glitch	xvpa	YD	Only positive motion for **vpa/mvpa** caps
semi_auto_right_margin	sam	YE	Printing in last column causes **cr**
status_line_esc_ok	eslok	es	Escape can be used on the status line
dest_tabs_magic_smso	xt	xt	Destructive tabs, magic **smso** char (t1061)
tilde_glitch	hz	hz	Hazeltine; can't print tilde (~)
transparent_underline	ul	ul	Underline character overstrikes
xon_xoff	xon	xo	Terminal uses xon/xoff handshaking

Numbers

Variable	Cap-name	Termcap Code	Description
buffer_capacity	bufsz	Ya	Number of bytes buffered before printing
columns	cols	co	Number of columns in a line
dot_vert_spacing	spinv	Yb	Spacing of pins vertically in pins per inch
dot_horz_spacing	spinh	Yc	Spacing of dots horizontally in dots per inch
init_tabs	it	it	Tabs initially every # spaces
label_height	lh	lh	Number of rows in each label
label_width	lw	lw	Number of columns in each label
lines	lines	li	Number of lines on a screen or a page
lines_of_memory	lm	lm	Lines of memory if > **lines**; **0** means varies
magic_cookie_glitch	xmc	sg	Number of blank characters left by **smso** or **rmso**
max_attributes	ma	ma	Maximum combined video attributes terminal can display
max_colors	colors	Co	Maximum number of colors on the screen
max_micro_address	maddr	Yd	Maximum value in **micro_..._address**
max_micro_jump	mjump	Ye	Maximum value in **parm_..._micro**
max_pairs	pairs	pa	Maximum number of color-pairs on the screen
maximum_windows	wnum	MW	Maximum number of definable windows
micro_col_size	mcs	Yf	Character step size when in micro mode
micro_line_size	mls	Yg	Line step size when in micro mode
no_color_video	ncv	NC	Video attributes that can't be used with colors
number_of_pins	npins	Yh	Number of pins in print-head
num_labels	nlab	Nl	Number of labels on screen (start at 1)
output_res_char	orc	Yi	Horizontal resolution in units per character
output_res_line	orl	Yj	Vertical resolution in units per line
output_res_horz_inch	orhi	Yk	Horizontal resolution in units per inch
output_res_vert_inch	orvi	Yl	Vertical resolution in units per inch
padding_baud_rate	pb	pb	Lowest baud rate where padding needed
print_rate	cps	Ym	Print rate in characters per second
virtual_terminal	vt	vt	Virtual terminal number (UNIX system)
wide_char_size	widcs	Yn	Character step size when in double wide mode
width_status_line	wsl	ws	Number of columns in status line

Strings

Variable	Cap-name	Termcap Code	Description
acs_chars	acsc	ac	Graphic charset pairs aAbBcC - def=vt100
back_tab	cbt	bt	Back tab
bell	bel	bl	Audible signal (bell)
carriage_return	cr	cr	Carriage return (*)
change_char_pitch	cpi	ZA	Change number of characters per inch (dg
change_line_pitch	lpi	ZB	Change number of lines per inch (dg
change_res_horz	chr	ZC	Change horizontal resolution (dg
change_res_vert	cvr	ZD	Change vertical resolution (dg
change_scroll_region	csr	cs	Change to lines #1 through #2 (vt100) (G)
char_padding	rmp	rP	Like **ip** but when in replace mode
char_set_names	csnm	Zy	List of character set names
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear all margins (top, bottom, and sides)
clear_screen	clear	cl	Clear screen and home cursor (*)
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display (*)
column_address	hpa	ch	Horizontal position absolute (G)
command_character	cmdch	CC	Terminal settable cmd character in prototype
create_window	cwin	CW	Define win #1 to go from #2,#3 to #4,#5
cursor_address	cup	cm	Move to row #1 col #2 (G)
cursor_down	cud1	do	Down one line
cursor_home	home	ho	Home cursor (if no **cup**)
cursor_invisible	civis	vi	Make cursor invisible
cursor_left	cub1	le	Move left one space.
cursor_mem_address	mrcup	CM	Memory relative cursor addressing (G)
cursor_normal	cnorm	ve	Make cursor appear normal (undo **vs/vi**)
cursor_right	cuf1	nd	Non-destructive space (cursor or carriage right)
cursor_to_ll	ll	ll	Last line, first column (if no **cup**)
cursor_up	cuu1	up	Upline (cursor up)
cursor_visible	cvvis	vs	Make cursor very visible
define_char	defc	ZE	Define a character in a character set †
delete_character	dch1	dc	Delete character (*)
delete_line	dl1	dl	Delete line (*)
delete_phone	dial	DI	Dial phone number #1
dis_status_line	dsl	ds	Disable status line
display_clock	dclk	DK	Display time-of-day clock
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate character set
enter_alt_charset_mode	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins
enter_blink_mode	blink	mb	Turn on blinking
enter_bold_mode	bold	md	Turn on bold (extra bright) mode
enter_ca_mode	smcup	ti	String to begin programs that use **cup**
enter_delete_mode	smdc	dm	Delete mode (enter)

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
enter_dim_mode	dim	mh	Turn on half-bright mode
enter_doublewide_mode	swidm	ZF	Enable double wide printing
enter_draft_quality	sdrfq	ZG	Set draft quality print
enter_insert_mode	smir	im	Insert mode (enter)
enter_italics_mode	sitm	ZH	Enable italics
enter_leftward_mode	slm	ZI	Enable leftward carriage motion
enter_micro_mode	smicm	ZJ	Enable micro motion capabilities
enter_near_letter_quality	snlq	ZK	Set near-letter quality print
enter_normal_quality	snrmq	ZL	Set normal quality print
enter_protected_mode	prot	mp	Turn on protected mode
enter_reverse_mode	rev	mr	Turn on reverse video mode
enter_secure_mode	invis	mk	Turn on blank mode (characters invisible)
enter_shadow_mode	sshm	ZM	Enable shadow printing
enter_standout_mode	smso	so	Begin standout mode
enter_subscript_mode	ssubm	ZN	Enable subscript printing
enter_superscript_mode	ssupm	ZO	Enable superscript printing
enter_underline_mode	smul	us	Start underscore mode
enter_upward_mode	sum	ZP	Enable upward carriage motion
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking
erase_chars	ech	ec	Erase #1 characters (G)
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
exit_attribute_mode	sgr0	me	Turn off all attributes
exit_ca_mode	rmcup	te	String to end programs that use **cup**
exit_delete_mode	rmdc	ed	End delete mode
exit_doublewide_mode	rwidm	ZQ	Disable double wide printing
exit_insert_mode	rmir	ei	End insert mode
exit_italics_mode	ritm	ZR	Disable italics
exit_leftward_mode	rlm	ZS	Enable rightward (normal) carriage motion
exit_micro_mode	rmicm	ZT	Disable micro motion capabilities
exit_shadow_mode	rshm	ZU	Disable shadow printing
exit_standout_mode	rmso	se	End standout mode
exit_subscript_mode	rsubm	ZV	Disable subscript printing
exit_superscript_mode	rsupm	ZW	Disable superscript printing
exit_underline_mode	rmul	ue	End underscore mode
exit_upward_mode	rum	ZX	Enable downward (normal) carriage motion
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking
fixed_pause	pause	PA	Pause for 2-3 seconds
flash_hook	hook	fh	Flash the switch hook
flash_screen	flash	vb	Visible bell (may not move cursor)
form_feed	ff	ff	Hardcopy terminal page eject (*)
from_status_line	fsl	fs	Return from status line
goto_window	wingo	WG	Got to window #1
hangup	hup	HU	Hang-up phone

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
init_1string	is1	i1	Terminal or printer initialization string
init_2string	is2	is	Terminal or printer initialization string
init_3string	is3	i3	Terminal or printer initialization string
init_file	if	if	Name of initialization file
init_prog	iprog	iP	Path name of program for initialization
initialize_color	initc	Ic	Initialize the definition of color
initialize_pair	initp	Ip	Initialize color-pair
insert_character	ich1	ic	Insert character
insert_line	il1	al	Add new blank line (*)
insert_padding	ip	ip	Insert pad after character inserted (*)
key_a1	ka1	K1	KEY_A1, 0534, upper left of keypad
key_a3	ka3	K3	KEY_A3, 0535, upper right of keypad
key_b2	kb2	K2	KEY_B2, 0536, center of keypad
key_backspace	kbs	kb	KEY_BACKSPACE, 0407, sent by backspace key
key_beg	kbeg	@1	KEY_BEG, 0542, sent by beg(inning) key
key_btab	kcbt	kB	KEY_BTAB, 0541, sent by back-tab key
key_c1	kc1	K4	KEY_C1, 0537, lower left of keypad
key_c3	kc3	K5	KEY_C3, 0540, lower right of keypad
key_cancel	kcan	@2	KEY_CANCEL, 0543, sent by cancel key
key_catab	ktbc	ka	KEY_CATAB, 0526, sent by clear-all-tabs key
key_clear	kclr	kC	KEY_CLEAR, 0515, sent by clear-screen or erase key
key_close	kclo	@3	KEY_CLOSE, 0544, sent by close key
key_command	kcmd	@4	KEY_COMMAND, 0545, sent by cmd (command) key
key_copy	kcpy	@5	KEY_COPY, 0546, sent by copy key
key_create	kcrt	@6	KEY_CREATE, 0547, sent by create key
key_ctab	kctab	kt	KEY_CTAB, 0525, sent by clear-tab key
key_dc	kdch1	kD	KEY_DC, 0512, sent by delete-character key
key_dl	kdl1	kL	KEY_DL, 0510, sent by delete-line key
key_down	kcud1	kd	KEY_DOWN, 0402, sent by terminal down-arrow key
key_eic	krmir	kM	KEY_EIC, 0514, sent by **rmir** or **smir** in insert mode
key_end	kend	@7	KEY_END, 0550, sent by end key
key_enter	kent	@8	KEY_ENTER, 0527, sent by enter/send key
key_eol	kel	kE	KEY_EOL, 0517, sent by clear-to-end-of-line key
key_eos	ked	kS	KEY_EOS, 0516, sent by clear-to-end-of-screen key
key_exit	kext	@9	KEY_EXIT, 0551, sent by exit key
key_f0	kf0	k0	KEY_F(0), 0410, sent by function key f0
key_f1	kf1	k1	KEY_F(C), 0411, sent by function key f1
key_f2	kf2	k2	KEY_F(S), 0412, sent by function key f2
key_f3	kf3	k3	KEY_F(S), 0413, sent by function key f3
key_f4	kf4	k4	KEY_F(F), 0414, sent by function key f4
key_f5	kf5	k5	KEY_F(M), 0415, sent by function key f5
key_f6	kf6	k6	KEY_F(6), 0416, sent by function key f6
key_f7	kf7	k7	KEY_F(7), 0417, sent by function key f7
key_f8	kf8	k8	KEY_F(8), 0420, sent by function key f8

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
key_f9	kf9	k9	KEY_F(9), 0421, sent by function key f9
key_f10	kf10	k;	KEY_F(ADM), 0422, sent by function key f10
key_f11	kf11	F1	KEY_F(ADM), 0423, sent by function key f11
key_f12	kf12	F2	KEY_F(ADM), 0424, sent by function key f12
key_f13	kf13	F3	KEY_F(ADM), 0425, sent by function key f13
key_f14	kf14	F4	KEY_F(ADM), 0426, sent by function key f14
key_f15	kf15	F5	KEY_F(ADM), 0427, sent by function key f15
key_f16	kf16	F6	KEY_F(ADM), 0430, sent by function key f16
key_f17	kf17	F7	KEY_F(ADM), 0431, sent by function key f17
key_f18	kf18	F8	KEY_F(ADM), 0432, sent by function key f18
key_f19	kf19	F9	KEY_F(ADM), 0433, sent by function key f19
key_f20	kf20	FA	KEY_F(20), 0434, sent by function key f20
key_f21	kf21	FB	KEY_F(21), 0435, sent by function key f21
key_f22	kf22	FC	KEY_F(22), 0436, sent by function key f22
key_f23	kf23	FD	KEY_F(23), 0437, sent by function key f23
key_f24	kf24	FE	KEY_F(24), 0440, sent by function key f24
key_f25	kf25	FF	KEY_F(25), 0441, sent by function key f25
key_f26	kf26	FG	KEY_F(26), 0442, sent by function key f26
key_f27	kf27	FH	KEY_F(27), 0443, sent by function key f27
key_f28	kf28	FI	KEY_F(28), 0444, sent by function key f28
key_f29	kf29	FJ	KEY_F(29), 0445, sent by function key f29
key_f30	kf30	FK	KEY_F(S), 0446, sent by function key f30
key_f31	kf31	FL	KEY_F(S), 0447, sent by function key f31
key_f32	kf32	FM	KEY_F(S), 0450, sent by function key f32
key_f33	kf33	FN	KEY_F(ADM), 0451, sent by function key f13
key_f34	kf34	FO	KEY_F(S), 0452, sent by function key f34
key_f35	kf35	FP	KEY_F(S), 0453, sent by function key f35
key_f36	kf36	FQ	KEY_F(S), 0454, sent by function key f36
key_f37	kf37	FR	KEY_F(S), 0455, sent by function key f37
key_f38	kf38	FS	KEY_F(S), 0456, sent by function key f38
key_f39	kf39	FT	KEY_F(S), 0457, sent by function key f39
key_f40	kf40	FU	KEY_F(40), 0460, sent by function key f40
key_f41	kf41	FV	KEY_F(41), 0461, sent by function key f41
key_f42	kf42	FW	KEY_F(42), 0462, sent by function key f42
key_f43	kf43	FX	KEY_F(43), 0463, sent by function key f43
key_f44	kf44	FY	KEY_F(44), 0464, sent by function key f44
key_f45	kf45	FZ	KEY_F(45), 0465, sent by function key f45
key_f46	kf46	Fa	KEY_F(46), 0466, sent by function key f46
key_f47	kf47	Fb	KEY_F(47), 0467, sent by function key f47
key_f48	kf48	Fc	KEY_F(48), 0470, sent by function key f48
key_f49	kf49	Fd	KEY_F(49), 0471, sent by function key f49
key_f50	kf50	Fe	KEY_F(50), 0472, sent by function key f50
key_f51	kf51	Ff	KEY_F(51), 0473, sent by function key f51

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
key_f52	kf52	Fg	KEY_F(52), 0474, sent by function key f52
key_f53	kf53	Fh	KEY_F(53), 0475, sent by function key f53
key_f54	kf54	Fi	KEY_F(54), 0476, sent by function key f54
key_f55	kf55	Fj	KEY_F(55), 0477, sent by function key f55
key_f56	kf56	Fk	KEY_F(56), 0500, sent by function key f56
key_f57	kf57	Fl	KEY_F(57), 0501, sent by function key f57
key_f58	kf58	Fm	KEY_F(58), 0502, sent by function key f58
key_f59	kf59	Fn	KEY_F(59), 0503, sent by function key f59
key_f60	kf60	Fo	KEY_F(60), 0504, sent by function key f60
key_f61	kf61	Fp	KEY_F(61), 0505, sent by function key f61
key_f62	kf62	Fq	KEY_F(62), 0506, sent by function key f62
key_f63	kf63	Fr	KEY_F(63), 0507, sent by function key f63
key_find	kfnd	@0	KEY_FIND, 0552, sent by find key
key_help	khlp	%1	KEY_HELP, 0553, sent by help key
key_home	khome	kh	KEY_HOME, 0406, sent by home key
key_ic	kich1	kI	KEY_IC, 0513, sent by ins-char/enter ins-mode key
key_il	kil1	kA	KEY_IL, 0511, sent by insert-line key
key_left	kcub1	kl	KEY_LEFT, 0404, sent by terminal left-arrow key
key_ll	kll	kH	KEY_LL, 0533, sent by home-down key
key_mark	kmrk	%2	KEY_MARK, 0554, sent by mark key
key_message	kmsg	%3	KEY_MESSAGE, 0555, sent by message key
key_move	kmov	%4	KEY_MOVE, 0556, sent by move key
key_next	knxt	%5	KEY_NEXT, 0557, sent by next key
key_npage	knp	kN	KEY_NPAGE, 0522, sent by next-page key
key_open	kopn	%6	KEY_OPEN, 0560, sent by open key
key_options	kopt	%7	KEY_OPTIONS, 0561, sent by options key
key_ppage	kpp	kP	KEY_PPAGE, 0523, sent by previous-page key
key_previous	kprv	%8	KEY_PREVIOUS, 0562, sent by pervious-object key
key_print	kprt	%9	KEY_PRINT, 0532, sent by print or copy key
key_redo	krdo	0	KEY_REDO, 0563, sent by redo key
key_reference	kref	&1	KEY_REFERENCE, 0564, sent by ref(erence) key
key_refresh	krfr	&2	KEY_REFRESH, 0565, sent by refresh key
key_replace	krpl	&3	KEY_REPLACE, 0566, sent by replace key
key_restart	krst	&4	KEY_RESTART, 0567, sent by restart key
key_resume	kres	&5	KEY_RESUME, 0570, sent by resume key
key_right	kcuf1	kr	KEY_RIGHT, 0405, sent by terminal right-arrow key
key_save	ksav	&6	KEY_SAVE, 0571, sent by save key
key_sbeg	kBEG	&9	KEY_SBEG, 0572, sent by shifted beginning key
key_scancel	kCAN	&0	KEY_SCANCEL, 0573, sent by shifted cancel key
key_scommand	kCMD	*1	KEY_SCOMMAND, 0574, sent by shifted command key
key_scopy	kCPY	*2	KEY_SCOPY, 0575, sent by shifted copy key
key_screate	kCRT	*3	KEY_SCREATE, 0576, sent by shifted create key
key_sdc	kDC	*4	KEY_SDC, 0577, sent by shifted delete-char key

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
key_sdl	kDL	*5	KEY_SDL, 0600, sent by shifted delete-line key
key_select	kslt	*6	KEY_SELECT, 0601, sent by select key
key_send	kEND	*7	KEY_SEND, 0602, sent by shifted end key
key_seol	kEOL	*8	KEY_SEOL, 0603, sent by shifted clear-line key
key_sexit	kEXT	*9	KEY_SEXIT, 0604, sent by shifted exit key
key_sf	kind	kF	KEY_SF, 0520, sent by scroll-forward/down key
key_sfind	kFND	*0	KEY_SFIND, 0605, sent by shifted find key
key_shelp	kHLP	#1	KEY_SHELP, 0606, sent by shifted help key
key_shome	kHOM	#2	KEY_SHOME, 0607, sent by shifted home key
key_sic	kIC	#3	KEY_SIC, 0610, sent by shifted input key
key_sleft	kLFT	#4	KEY_SLEFT, 0611, sent by shifted left-arrow key
key_smessage	kMSG	%a	KEY_SMESSAGE, 0612, sent by shifted message key
key_smove	kMOV	%b	KEY_SMOVE, 0613, sent by shifted move key
key_snext	kNXT	%c	KEY_SNEXT, 0614, sent by shifted next key
key_soptions	kOPT	%d	KEY_SOPTIONS, 0615, sent by shifted options key
key_sprevious	kPRV	%e	KEY_SPREVIOUS, 0616, sent by shifted prev key
key_sprint	kPRT	%f	KEY_SPRINT, 0617, sent by shifted print key
key_sr	kri	kR	KEY_SR, 0521, sent by scroll-backward/up key
key_sredo	kRDO	%g	KEY_SREDO, 0620, sent by shifted redo key
key_sreplace	kRPL	%h	KEY_SREPLACE, 0621, sent by shifted replace key
key_sright	kRIT	%i	KEY_SRIGHT, 0622, sent by shifted right-arrow key
key_srsume	kRES	%j	KEY_SRSUME, 0623, sent by shifted resume key
key_ssave	kSAV	!1	KEY_SSAVE, 0624, sent by shifted save key
key_ssuspend	kSPD	!2	KEY_SSUSPEND, 0625, sent by shifted suspend key
key_stab	khts	kT	KEY_STAB, 0524, sent by set-tab key
key_sundo	kUND	!3	KEY_SUNDO, 0626, sent by shifted undo key
key_suspend	kspd	&7	KEY_SUSPEND, 0627, sent by suspend key
key_undo	kund	&8	KEY_UNDO, 0630, sent by undo key
key_up	kcuu1	ku	KEY_UP, 0403, sent by terminal up-arrow key
keypad_local	rmkx	ke	Out of ''keypad-transmit'' mode
keypad_xmit	smkx	ks	Put terminal in ''keypad-transmit'' mode
lab_f0	lf0	l0	Labels on function key f0 if not f0
lab_f1	lf1	l1	Labels on function key f1 if not f1
lab_f2	lf2	l2	Labels on function key f2 if not f2
lab_f3	lf3	l3	Labels on function key f3 if not f3
lab_f4	lf4	l4	Labels on function key f4 if not f4
lab_f5	lf5	l5	Labels on function key f5 if not f5
lab_f6	lf6	l6	Labels on function key f6 if not f6
lab_f7	lf7	l7	Labels on function key f7 if not f7
lab_f8	lf8	l8	Labels on function key f8 if not f8
lab_f9	lf9	l9	Labels on function key f9 if not f9
lab_f10	lf10	la	Labels on function key f10 if not f10
label_format	fln	Lf	Label format

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
label_off	rmln	LF	Turn off soft labels
label_on	smln	LO	Turn on soft labels
meta_off	rmm	mo	Turn off "meta mode"
meta_on	smm	mm	Turn on "meta mode" (8th bit)
micro_column_address	mhpa	ZY	Like **column_address** for micro adjustment †
micro_down	mcud1	ZZ	Like **cursor_down** for micro adjustment
micro_left	mcub1	Za	Like **cursor_left** for micro adjustment
micro_right	mcuf1	Zb	Like **cursor_right** for micro adjustment
micro_row_address	mvpa	Zc	Like **row_address** for micro adjustment †
micro_up	mcuu1	Zd	Like **cursor_up** for micro adjustment
newline	nel	nw	Newline (behaves like **cr** followed by **lf**)
order_of_pins	porder	Ze	Matches software bits to print-head pins
orig_colors	oc	oc	Set all color(-pair)s to the original ones
orig_pair	op	op	Set default color-pair to the original one
pad_char	pad	pc	Pad character (rather than null)
parm_dch	dch	DC	Delete #1 chars (G*)
parm_delete_line	dl	DL	Delete #1 lines (G*)
parm_down_cursor	cud	DO	Move down #1 lines. (G*)
parm_down_micro	mcud	Zf	Like **parm_down_cursor** for micro adjust. (G*)
parm_ich	ich	IC	Insert #1 blank chars (G*)
parm_index	indn	SF	Scroll forward #1 lines. (G)
parm_insert_line	il	AL	Add #1 new blank lines (G*)
parm_left_cursor	cub	LE	Move cursor left #1 spaces (G)
parm_left_micro	mcub	Zg	Like **parm_left_cursor** for micro adjust. †
parm_right_cursor	cuf	RI	Move right #1 spaces. (G*)
parm_right_micro	mcuf	Zh	Like **parm_right_cursor** for micro adjust. †
parm_rindex	rin	SR	Scroll backward #1 lines. (G)
parm_up_cursor	cuu	UP	Move cursor up #1 lines. (G*)
parm_up_micro	mcuu	Zi	Like **parm_up_cursor** for micro adjust. †
pkey_key	pfkey	pk	Prog funct key #1 to type string #2
pkey_local	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit	pfx	px	Prog funct key #1 to xmit string #2
plab_norm	pln	pn	Prog label #1 to show string #2
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	pO	Turn on the printer for #1 bytes
prtr_off	mc4	pf	Turn off the printer
prtr_on	mc5	po	Turn on the printer
pulse	pulse	PU	Select pulse dialing
quick_dial	qdial	QD	Dial phone number #1, without progress detection
remove_clock	rmclk	RC	Remove time-of-day clock
repeat_char	rep	rp	Repeat char #1 #2 times (G*)
req_for_input	rfi	RF	Send next input char (for ptys)

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
reset_1string	rs1	r1	Reset terminal completely to sane modes
reset_2string	rs2	r2	Reset terminal completely to sane modes
reset_3string	rs3	r3	Reset terminal completely to sane modes
reset_file	rf	rf	Name of file containing reset string
restore_cursor	rc	rc	Restore cursor to position of last sc
row_address	vpa	cv	Vertical position absolute (G)
save_cursor	sc	sc	Save cursor position
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	sr	Scroll text down
select_char_set	scs	Zj	Select character set †
set_attributes	sgr	sa	Define the video attributes (G) #1-#9
set_background	setb	Sb	Set current background color
set_bottom_margin	smgb	Zk	Set bottom margin at current line
set_bottom_margin_parm	smgbp	Zl	Set bottom margin at line #1 †
set_clock	sclk	SC	Set time-of-day clock
set_color_pair	scp	sp	Set current color-pair
set_foreground	setf	Sf	Set current foreground color1
set_left_margin	smgl	ML	Set left margin at current line
set_left_margin_parm	smglp	Zm	Set left margin at column #1 †
set_right_margin	smgr	MR	Set right margin at current column
set_right_margin_parm	smgrp	Zn	Set right margin at column #1 †
set_tab	hts	st	Set a tab in all rows, current column
set_top_margin	smgt	Zo	Set top margin at current line
set_top_margin_parm	smgtp	Zp	Set top margin at line #1 †
set_window	wind	wi	Current window is lines #1-#2 cols #3-#4 (G)
start_bit_image	sbim	Zq	Start printing bit image graphics †
start_char_set_def	scsd	Zr	Start definition of a character set †
stop_bit_image	rbim	Zs	End printing bit image graphics
stop_char_set_def	rcsd	Zt	End definition of a character set
subscript_characters	subcs	Zu	List of "subscript-able" characters
superscript_characters	supcs	Zv	List of "superscript-able" characters
tab	ht	ta	Tab to next 8-space hardware tab stop
these_cause_cr	docr	Zw	Printing any of these chars causes **cr**
to_status_line	tsl	ts	Go to status line, col #1 (G)
tone	tone	TO	Select touch tone dialing
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
user0	u0	u0	User string 0
user1	u1	u1	User string 1
user2	u2	u2	User string 4
user3	u3	u3	User string 3
user4	u4	u4	User string 4

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
user5	u5	u5	User string 5
user6	u6	u6	User string 6
user7	u7	u7	User string 7
user8	u8	u8	User string 8
user9	u9	u9	User string 9
wait_tone	wait	WA	Wait for dial tone
xoff_character	xoffc	XF	X-off character
xon_character	xonc	XN	X-on character
zero_motion	zerom	Zx	No motion for the subsequent character

Booleans

Cap-name	Variable	Termcap Code	Description
am	auto_right_margin	am	Terminal has automatic margins
bw	auto_left_margin	bw	**cub1** wraps from column 0 to last column
ccc	can_change	cc	Terminal can re-define existing color
chts	hard_cursor	HC	Cursor is hard to see
cpix	cpi_changes_res	YF	Changing character pitch changes resolution
cps	print_rate	Ym	Print rate in characters per second
crxm	cr_cancels_micro_modem	YB	Using **cr** turns off micro mode
cwin	create_window	CW	Define win #1 to go from #2,#3 to #4,#5
da	memory_above	da	Display may be retained above the screen
daisy	has_print_wheel	YC	Printer needs operator to change character set
dclk	display_clock	DK	Display time-of-day clock
db	memory_below	db	Display may be retained below the screen
dial	dial_phone	DI	Dial phone number #1
eo	erase_overstrike	eo	Can erase overstrikes with a blank
eslok	status_line_esc_ok	es	Escape can be used on the status line
gn	generic_type	gn	Generic line type (e.g., dialup, switch)
hc	hard_copy	hc	Hardcopy terminal
hls	hue_lightness_saturation	hl	Terminal uses only HLS color notation (Tektronix)
hs	has_status_line	hs	Has extra "status line"
hz	tilde_glitch	hz	Hazeltine; can't print tilde (~)
in	insert_null_glitch	in	Insert mode distinguishes nulls
km	has_meta_key	km	Has a meta key (shift, sets parity bit)
lpix	lpi_changes_res	YG	Changing line pitch changes resolution
mc5i	prtr_silent		
mir	move_insert_mode	mi	Safe to move while in insert mode
msgr	move_standout_mode	ms	Safe to move in standout modes
npc	no_pad_char	NP	Pad character doesn't exist
nrrmc	non_rev_rmcup	NR	**smcup** does not reverse **rmcup**
nxon	needs_xon_xoff	nx	Padding won't work, xon/xoff required
os	over_strike	os	Terminal overstrikes on hard-copy terminal
sam	semi_auto_right_margin	YE	Printing in last column causes **cr**
ul	transparent_underline	ul	Underline character overstrikes
xenl	eat_newline_glitch	xn	Newline ignored after 80 columns (*Concept*)
xhp	ceol_standout_glitch	xs	Standout not erased by overwriting (hp)
xhpa	col_addr_glitch	YA	Only positive motion for **hpa/mhpa** caps
xon	xon_xoff	xo	Terminal uses xon/xoff handshaking
xsb	no_esc_ctlc	xb	Beehive (f1=escape, f2=ctrl C)
xt	dest_tabs_magic_smso	xt	Destructive tabs, magic **smso** char (t1061)
xvpa	row_addr_glitch	YD	Only positive motion for **vpa/mvpa** caps

Numbers

Cap-name	Variable	Termcap Code	Description
bufsz	buffer_capacity	Ya	Number of bytes buffered before printing
colors	max_colors	Co	Maximum number of colors on the screen
cols	columns	co	Number of columns in a line
cps	print_rate	Ym	Average print rate in characters per second
it	init_tabs	it	Tabs initially every # spaces
lh	label_height	lh	Number of rows in each label
lines	lines	li	Number of lines on a screen or a page
lm	lines_of_memory	lm	Lines of memory if > **lines**; **0** means varies
lw	label_width	lw	Number of columns in each label
maddr	max_micro_address	Yd	Maximum value in **micro_..._address**
mcs	micro_col_size	Yf	Character step size when in micro mode
mjump	max_micro_jump	Ye	Maximum value in **parm_..._micro**
mls	micro_line_size	Yg	Line step size when in micro mode
ncv	no_color_video	NC	Video attributes that can't be used with colors
nlab	num_labels	Nl	Number of labels on screen (start at 1)
npins	number_of_pins	Yh	Number of pins in print-head
orc	output_res_char	Yi	Horizontal resolution in units per character
orhi	output_res_horz_inch	Yk	Horizontal resolution in units per inch
orl	output_res_line	Yj	Vertical resolution in units per line
orvi	output_res_vert_inch	Yl	Vertical resolution in units per inch
pairs	max_pairs	pa	Maximum number of color-pairs on the screen
pb	padding_baud_rate	pb	Lowest baud rate where padding needed
spinh	dot_horz_spacing	Yc	Spacing of dots horizontally in dots per inch
spinv	dot_vert_spacing	Yb	Spacing of pins vertically in pins per inch
vt	virtual_terminal	vt	Virtual terminal number (UNIX system)
widcs	wide_char_size	Yn	Character step size when in double wide mode
wsl	width_status_line	ws	Number of columns in status line
xmc	magic_cookie_glitch	sg	Number of blank characters left by **smso** or **rmso**

Strings

Cap-name	Variable	Termcap Code	Description
acsc	acs_chars	ac	Graphic charset pairs aAbBcC - def=vt100
bel	bell	bl	Audible signal (bell)
blink	enter_blink_mode	mb	Turn on blinking
bold	enter_bold_mode	md	Turn on bold (extra bright) mode
cbt	back_tab	bt	Back tab
chr	change_res_horz	ZC	Change horizontal resolution †
civis	cursor_invisible	vi	Make cursor invisible
clear	clear_screen	cl	Clear screen and home cursor (*)
cmdch	command_character	CC	Terminal settable cmd character in prototype
cnorm	cursor_normal	ve	Make cursor appear normal (undo vs/vi)
cpi	change_char_pitch	ZA	Change number of characters per inch †
cr	carriage_return	cr	Carriage return (*)
csnm	char_set_names	Zy	List of character set names
csr	change_scroll_region	cs	Change to lines #1 through #2 (vt100) (G)
cub	parm_left_cursor	LE	Move cursor left #1 spaces (G)
cub1	cursor_left	le	Move left one space.
cud	parm_down_cursor	DO	Move down #1 lines. (G*)
cuf	parm_right_cursor	RI	Move right #1 spaces. (G*)
cuf1	cursor_right	nd	Non-destructive space (cursor or carriage right)
cup	cursor_address	cm	Move to row #1 col #2 (G)
cuu	parm_up_cursor	UP	Move cursor up #1 lines. (G*)
cvr	change_res_vert	ZD	Change vertical resolution †
cvvis	cursor_visible	vs	Make cursor very visible
dch	parm_dch	DC	Delete #1 chars (G*)
dch1	delete_character	dc	Delete character (*)
defc	define_char	ZE	Define a character in a character set
dim	enter_dim_mode	mh	Turn on half-bright mode
dl	delete_line	dl1	Delete line (*)
dl	parm_delete_line	DL	Delete #1 lines (G*)
do	cursor_down	do	Down one line
docr	these_cause_cr	Zw	Printing any of these chars causes **cr**
dsl	dis_status_line	ds	Disable status line
ech	erase_chars	ec	Erase #1 characters (G)
ed	clr_eos	cd	Clear to end of display (*)
el	clr_eol	ce	Clear to end of line
el1	clr_bol	cb	Clear to beginning of line, inclusive
enacs	ena_acs	eA	Enable alternate character set
ff	form_feed	ff	Hardcopy terminal page eject (*)
flash	flash_screen	vb	Visible bell (may not move cursor)
fln	label_format	Lf	Label format
fsl	from_status_line	fs	Return from status line
hd	down_half_line	hd	Half-line down (forward 1/2 linefeed)
home	cursor_home	ho	Home cursor (if no **cup**)

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
hook	flash_hook	fh	Flash the switch hook
hpa	column_address	ch	Horizontal position absolute (G)
ht	tab	ta	Tab to next 8-space hardware tab stop
hts	set_tab	st	Set a tab in all rows, current column
hu	up_half_line	hu	Half-line up (reverse 1/2 linefeed)
hup	hangup	HU	Hang-up phone
ich	parm_ich	IC	Insert #1 blank chars (G*)
ich1	insert_character	ic	Insert character
if	init_file	if	Name of initialization file
il	parm_insert_line	AL	Add #1 new blank lines (G*)
il1	insert_line	al	Add new blank line (*)
ind	scroll_forward	sf	Scroll text up
indn	parm_index	SF	Scroll forward #1 lines. (G)
initc	initialize_color	Ic	Initialize the definition of color
initp	initialize_pair	Ip	Initialize color-pair
invis	enter_secure_mode	mk	Turn on blank mode (characters invisible)
ip	insert_padding	ip	Insert pad after character inserted (*)
iprog	init_prog	iP	Path name of program for initialization
is1	init_1string	i1	Terminal or printer initialization string
is2	init_2string	is	Terminal or printer initialization string
is3	init_3string	i3	Terminal or printer initialization string
kBEG	key_sbeg	&9	KEY_SBEG, 0572, sent by shifted beginning key
kCAN	key_scancel	&0	KEY_SCANCEL, 0573, sent by shifted cancel key
kCMD	key_scommand	*1	KEY_SCOMMAND, 0574, sent by shifted command key
kCPY	key_scopy	*2	KEY_SCOPY, 0575, sent by shifted copy key
kCRT	key_screate	*3	KEY_SCREATE, 0576, sent by shifted create key
kDC	key_sdc	*4	KEY_SDC, 0577, sent by shifted delete-char key
kDL	key_sdl	*5	KEY_SDL, 0600, sent by shifted delete-line key
kEND	key_send	*7	KEY_SEND, 0602, sent by shifted end key
kEOL	key_seol	*8	KEY_SEOL, 0603, sent by shifted clear-line key
kEXT	key_sexit	*9	KEY_SEXIT, 0604, sent by shifted exit key
kFND	key_sfind	*0	KEY_SFIND, 0605, sent by shifted find key
kHLP	key_shelp	#1	KEY_SHELP, 0606, sent by shifted help key
kHOM	key_shome	#2	KEY_SHOME, 0607, sent by shifted home key
kIC	key_sic	#3	KEY_SIC, 0610, sent by shifted input key
kLFT	key_sleft	#4	KEY_SLEFT, 0611, sent by shifted left-arrow key
kMOV	key_smove	b	KEY_SMOVE, 0613, sent by shifted move key
kMSG	key_smessage	%a	KEY_SMESSAGE, 0612, sent by shifted message key
kNXT	key_snext	%c	KEY_SNEXT, 0614, sent by shifted next key
kOPT	key_soptions	%d	KEY_SOPTIONS, 0615, sent by shifted options key
kPRT	key_sprint	%f	KEY_SPRINT, 0617, sent by shifted print key
kPRV	key_sprevious	%e	KEY_SPREVIOUS, 0616, sent by shifted prev key
kRDO	key_sredo	%g	KEY_SREDO, 0620, sent by shifted redo key
kRES	key_srsume	%j	KEY_SRSUME, 0623, sent by shifted resume key

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
kRIT	key_sright	%i	KEY_SRIGHT, 0622, sent by shifted right-arrow key
kRPL	key_sreplace	%h	KEY_SREPLACE, 0621, sent by shifted replace key
kSAV	key_ssave	!1	KEY_SSAVE, 0624, sent by shifted save key
kSPD	key_ssuspend	!2	KEY_SSUSPEND, 0625, sent by shifted suspend key
kUND	key_sundo	!3	KEY_SUNDO, 0626, sent by shifted undo key
ka1	key_a1	K1	KEY_A1, 0534, upper left of keypad
ka3	key_a3	K3	KEY_A3, 0535, upper right of keypad
kb2	key_b2	K2	KEY_B2, 0536, center of keypad
kbeg	key_beg	@1	KEY_BEG, 0542, sent by beg(inning) key
kbs	key_backspace	kb	KEY_BACKSPACE, 0407, sent by backspace key
kc1	key_c1	K4	KEY_C1, 0537, lower left of keypad
kc3	key_c3	K5	KEY_C3, 0540, lower right of keypad
kcan	key_cancel	@2	KEY_CANCEL, 0543, sent by cancel key
kcbt	key_btab	kB	KEY_BTAB, 0541, sent by back-tab key
kclo	key_close	@3	KEY_CLOSE, 0544, sent by close key
kclr	key_clear	kC	KEY_CLEAR, 0515, sent by clear-screen or erase key
kcmd	key_command	@4	KEY_COMMAND, 0545, sent by cmd (command) key
kcpy	key_copy	@5	KEY_COPY, 0546, sent by copy key
kcrt	key_create	@6	KEY_CREATE, 0547, sent by create key
kctab	key_ctab	kt	KEY_CTAB, 0525, sent by clear-tab key
kcub1	key_left	kl	KEY_LEFT, 0404, sent by terminal left-arrow key
kcud1	key_down	kd	KEY_DOWN, 0402, sent by terminal down-arrow key
kcuf1	key_right	kr	KEY_RIGHT, 0405, sent by terminal right-arrow key
kcuu1	key_up	ku	KEY_UP, 0403, sent by terminal up-arrow key
kdch1	key_dc	kD	KEY_DC, 0512, sent by delete-character key
kdl1	key_dl	kL	KEY_DL, 0510, sent by delete-line key
ked	key_eos	ked	KEY_EOS, 0516, sent by clear-to-end-of-screen key
kel	key_eol	kE	KEY_EOL, 0517, sent by clear-to-end-of-line key
kend	key_end	@7	KEY_END, 0550, sent by end kee
kent	key_enter	@8	KEY_ENTER, 0527, sent by enter/send key
kext	key_exit	@9	KEY_EXIT, 0551, sent by exit key
kf0	key_f0	k0	KEY_F(0), 0410, sent by function key f0
kf1	key_f1	k1	KEY_F(C), 0411, sent by function key f1
kf10	key_f10	k;	KEY_F(ADM), 0422, sent by function key f10
kf11	key_f11	F1	KEY_F(ADM), 0423, sent by function key f11
kf12	key_f12	F2	KEY_F(ADM), 0424, sent by function key f12
kf13	key_f13	F3	KEY_F(ADM), 0425, sent by function key f13
kf14	key_f14	F4	KEY_F(ADM), 0426, sent by function key f14
kf15	key_f15	F5	KEY_F(ADM), 0427, sent by function key f15
kf16	key_f16	F6	KEY_F(ADM), 0430, sent by function key f16
kf17	key_f17	F7	KEY_F(ADM), 0431, sent by function key f17
kf18	key_f18	F8	KEY_F(ADM), 0432, sent by function key f18
kf19	key_f19	F9	KEY_F(ADM), 0433, sent by function key f19
kf2	key_f2	k2	KEY_F(S), 0412, sent by function key f2

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
kf20	key_f20	FA	KEY_F(20), 0434, sent by function key f20
kf21	key_f21	FB	KEY_F(21), 0435, sent by function key f21
kf22	key_f22	FC	KEY_F(22), 0436, sent by function key f22
kf23	key_f23	FD	KEY_F(23), 0437, sent by function key f23
kf24	key_f24	FE	KEY_F(24), 0440, sent by function key f24
kf25	key_f25	FF	KEY_F(25), 0441, sent by function key f25
kf26	key_f26	FG	KEY_F(26), 0442, sent by function key f26
kf27	key_f27	FH	KEY_F(27), 0443, sent by function key f27
kf28	key_f28	FI	KEY_F(28), 0444, sent by function key f28
kf29	key_f29	FJ	KEY_F(29), 0445, sent by function key f29
kf3	key_f3	k3	KEY_F(S), 0413, sent by function key f3
kf30	key_f30	FK	KEY_F(S), 0446, sent by function key f30
kf31	key_f31	FL	KEY_F(S), 0447, sent by function key f31
kf32	key_f32	FM	KEY_F(S), 0450, sent by function key f32
kf33	key_f33	FN	KEY_F(ADM), 0451, sent by function key f13
kf34	key_f34	FO	KEY_F(S), 0452, sent by function key f34
kf35	key_f35	FP	KEY_F(S), 0453, sent by function key f35
kf36	key_f36	FQ	KEY_F(S), 0454, sent by function key f36
kf37	key_f37	FR	KEY_F(S), 0455, sent by function key f37
kf38	key_f38	FS	KEY_F(S), 0456, sent by function key f38
kf39	key_f39	FT	KEY_F(S), 0457, sent by function key f39
kf4	key_f4	k4	KEY_F(F), 0414, sent by function key f4
kf40	key_f40	FU	KEY_F(40), 0460, sent by function key f40
kf41	key_f41	FV	KEY_F(41), 0461, sent by function key f41
kf42	key_f42	FW	KEY_F(42), 0462, sent by function key f42
kf43	key_f43	FX	KEY_F(43), 0463, sent by function key f43
kf44	key_f44	FY	KEY_F(44), 0464, sent by function key f44
kf45	key_f45	FZ	KEY_F(45), 0465, sent by function key f45
kf46	key_f46	Fa	KEY_F(46), 0466, sent by function key f46
kf47	key_f47	Fb	KEY_F(47), 0467, sent by function key f47
kf48	key_f48	Fc	KEY_F(48), 0470, sent by function key f48
kf49	key_f49	Fd	KEY_F(49), 0471, sent by function key f49
kf5	key_f5	k5	KEY_F(M), 0415, sent by function key f5
kf50	key_f50	Fe	KEY_F(50), 0472, sent by function key f50
kf51	key_f51	Ff	KEY_F(51), 0473, sent by function key f51
kf52	key_f52	Fg	KEY_F(52), 0474, sent by function key f52
kf53	key_f53	Fh	KEY_F(53), 0475, sent by function key f53
kf54	key_f54	Fi	KEY_F(54), 0476, sent by function key f54
kf55	key_f55	Fj	KEY_F(55), 0477, sent by function key f55
kf56	key_f56	Fk	KEY_F(56), 0500, sent by function key f56
kf57	key_f57	Fl	KEY_F(57), 0501, sent by function key f57
kf58	key_f58	Fm	KEY_F(58), 0502, sent by function key f58
kf59	key_f59	Fn	KEY_F(59), 0503, sent by function key f59
kf6	key_f6	k6	KEY_F(6), 0416, sent by function key f6

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
kf60	key_f60	Fo	KEY_F(60), 0504, sent by function key f60
kf61	key_f61	Fp	KEY_F(61), 0505, sent by function key f61
kf62	key_f62	Fq	KEY_F(62), 0506, sent by function key f62
kf63	key_f63	Fr	KEY_F(63), 0507, sent by function key f63
kf7	key_f7	k7	KEY_F(7), 0417, sent by function key f7
kf8	key_f8	k8	KEY_F(8), 0420, sent by function key f8
kf9	key_f9	k9	KEY_F(9), 0421, sent by function key f9
kfnd	key_find	@0	KEY_FIND, 0552, sent by find key
khlp	key_help	%1	KEY_HELP, 0553, sent by help key
khome	key_home	kh	KEY_HOME, 0406, sent by home key
khts	key_stab	kT	KEY_STAB, 0524, sent by set-tab key
kich1	key_ic	kI	KEY_IC, 0513, sent by ins-char/enter ins-mode key
kil1	key_il	kA	KEY_IL, 0511, sent by insert-line key
kind	key_sf	kF	KEY_SF, 0520, sent by scroll-forward/down key
kll	key_ll	kH	KEY_LL, 0533, sent by home-down key
kmov	key_move	%4	KEY_MOVE, 0556, sent by move key
kmrk	key_mark	%2	KEY_MARK, 0554, sent by mark key
kmsg	key_message	%3	KEY_MESSAGE, 0555, sent by message key
knp	key_npage	kN	KEY_NPAGE, 0522, sent by next-page key
knxt	key_next	%5	KEY_NEXT, 0557, sent by next-object key
kopn	key_open	%6	KEY_OPEN, 0560, sent by open key
kopt	key_options	%7	KEY_OPTIONS, 0561, sent by options key
kpp	key_ppage	kP	KEY_PPAGE, 0523, sent by previous-page key
kprt	key_print	%9	KEY_PRINT, 0532, sent by print or copy key
kprv	key_previous	%8	KEY_PREVIOUS, 0562, sent by previous-object key
krdo	key_redo	%0	KEY_REDO, 0563, sent by redo key
kref	key_reference	&1	KEY_REFERENCE, 0564, sent by ref(erence) key
kres	key_resume	&5	KEY_RESUME, 0570, sent by resume key
krfr	key_refresh	&2	KEY_REFRESH, 0565, sent by refresh key
kri	key_sr	kR	KEY_SR, 0521, sent by scroll-backward/up key
krmir	key_eic	kM	KEY_EIC, 0514, sent by **rmir** or **smir** in insert mode
krpl	key_replace	&3	KEY_REPLACE, 0566, sent by replace key
krst	key_restart	&4	KEY_RESTART, 0567, sent by restart key
ksav	key_save	&6	KEY_SAVE, 0571, sent by save key
kslt	key_select	*6	KEY_SELECT, 0601, sent by select key
kspd	key_suspend	&7	KEY_SUSPEND, 0627, sent by suspend key
ktbc	key_catab	ka	KEY_CATAB, 0526, sent by clear-all-tabs key
kund	key_undo	&8	KEY_UNDO, 0630, sent by undo key
lf0	lab_f0	l0	Labels on function key f0 if not f0
lf1	lab_f1	l1	Labels on function key f1 if not f1
lf10	lab_f10	la	Labels on function key f10 if not f10
lf2	lab_f2	l2	Labels on function key f2 if not f2
lf3	lab_f3	l3	Labels on function key f3 if not f3
lf4	lab_f4	l4	Labels on function key f4 if not f4

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
lf5	lab_f5	l5	Labels on function key f5 if not f5
lf6	lab_f6	l6	Labels on function key f6 if not f6
lf7	lab_f7	l7	Labels on function key f7 if not f7
lf8	lab_f8	l8	Labels on function key f8 if not f8
lf9	lab_f9	l9	Labels on function key f9 if not f9
ll	cursor_to_ll	ll	Last line, first column (if no **cup**)
lpi	change_line_pitch	ZB	Change number of lines per inch †
ma	max_attributes	ma	Maximum combined video attributes terminal can display
mc0	print_screen	ps	Print contents of the screen
mc4	prtr_off	pf	Turn off the printer
mc5	prtr_on	po	Turn on the printer
mc5p	prtr_non	pO	Turn on the printer for #1 bytes
mcub	parm_left_micro	Zg	Like **parm_left_cursor** for micro adjust. †
mcub1	micro_left	Za	Like **cursor_left** for micro adjustment
mcud	parm_down_micro	Zf	Like **parm_down_cursor** for micro adjust. (G*)
mcud1	micro_down	ZZ	Like **cursor_down** for micro adjustment
mcuf	parm_right_micro	Zh	Like **parm_right_cursor** for micro adjust. †
mcuf1	micro_right	Zb	Like **cursor_right** for micro adjustment
mcuu	parm_up_micro	Zi	Like **parm_up_cursor** for micro adjust. †
mcuu1	micro_up	Zd	Like **cursor_up** for micro adjustment
mgc	clear_margins	MC	Clear all margins (top, bottom, and sides)
mhpa	micro_column_address	ZY	Like **column_address** for micro adjustment †
mrcup	cursor_mem_address	CM	Memory relative cursor addressing (G)
mvpa	micro_row_address	Zc	Like **row_address** for micro adjustment †
ndscr	non_dest_scroll_region	ND	Scrolling region is non-destructive
nel	newline	nw	Newline (behaves like **cr** followed by **lf**)
oc	orig_colors	oc	Set all color(-pair)s to the original ones
op	orig_pair	op	Set default color-pair to the original one
pad	pad_char	pc	Pad character (rather than null)
pause	fixed_pause	PA	Pause for 2-3 seconds
pfkey	pkey_key	pk	Prog funct key #1 to type string #2
pfloc	pkey_local	pl	Prog funct key #1 to execute string #2
pfx	pkey_xmit	px	Prog funct key #1 to xmit string #2
pln	plab_norm	pn	Prog label #1 to show string #2
porder	order_of_pins	Ze	Matches software bits to print-head pins
prot	enter_protected_mode	mp	Turn on protected mode
pulse	pulse	PU	Select pulse dialing
qdial	quick_dial	QD	Dial phone number #1, without progress detection
rbim	stop_bit_image	Zs	End printing bit image graphics
rc	restore_cursor	rc	Restore cursor to position of last sc
rcsd	stop_char_set_def	Zt	End definition of a character set
rep	repeat_char	rp	Repeat char #1 #2 times (G*)
rev	enter_reverse_mode	mr	Turn on reverse video mode
rf	reset_file	rf	Name of file containing reset string

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
rfi	req_for_input	RF	Send next input char (for ptys)
ri	scroll_reverse	sr	Scroll text down
rin	parm_rindex	SR	Scroll backward #1 lines. (G)
ritm	exit_italics_mode	ZR	Disable italics
rlm	exit_leftward_mode	ZS	Enable rightward (normal) carriage motion
rmacs	exit_alt_charset_mode	ae	End alternate character set
rmam	exit_am_mode	RA	Turn off automatic margins
rmclk	remove_clock	RC	Remove time-of-day clock
rmcup	exit_ca_mode	te	String to end programs that use **cup**
rmdc	exit_delete_mode	ed	End delete mode
rmicm	exit_micro_mode	ZT	Disable micro motion capabilities
rmir	exit_insert_mode	ei	End insert mode
rmkx	keypad_local	ke	Out of ''keypad-transmit'' modey
rmln	label_off	LF	Turn off soft labels
rmm	meta_off	mo	Turn off "meta mode"
rmp	char_padding	rP	Like **ip** but when in replace mode
rmso	exit_standout_mode	se	End standout mode
rmul	exit_underline_mode	ue	End underscore mode
rmxon	exit_xon_mode	RX	Turn off xon/xoff handshaking
rs1	reset_1string	r1	Reset terminal completely to sane modes
rs2	reset_2string	r2	Reset terminal completely to sane modes
rs3	reset_3string	r3	Reset terminal completely to sane modes
rshm	exit_shadow_mode	ZU	Disable shadow printing
rsubm	exit_subscript_mode	ZV	Disable subscript printing
rsupm	exit_superscript_mode	ZW	Disable superscript printing
rum	exit_upward_mode	ZX	Enable downward (normal) carriage motion
rwidm	exit_doublewide_mode	ZQ	Disable double wide printing
sbim	start_bit_image	Zq	Start printing bit image graphics †
sc	save_cursor	sc	Save cursor position
sclk	set_clock	SC	Set time-of-day clock
scp	set_color_pair	sp	Set current color-pair
scs	select_char_set	Zj	Select character set †
scsd	start_char_set_def	Zr	Start definition of a character set †
sdrfq	enter_draft_quality	ZG	Set draft quality print
setb	set_background	Sb	Set current background color
setf	set_foreground	Sf	Set current foreground color
sgr	set_attributes	sa	Define the video attributes #1-#9 (G)
sgr0	exit_attribute_mode	me	Turn off all attributes
sitm	enter_italics_mode	ZH	Enable italics
slm	enter_leftward_mode	ZI	Enable leftward carriage motion
smacs	enter_alt_charset_mode	as	Start alternate character set
smam	enter_am_mode	SA	Turn on automatic margins
smcup	enter_ca_mode	ti	String to begin programs that use **cup**
smdc	enter_delete_mode	dm	Delete mode (enter)

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
smgb	set_bottom_margin	Zk	Set bottom margin at current line
smgbp	set_bottom_margin_parm	Zl	Set bottom margin at line #1 †
smgl	set_left_margin	ML	Set left margin at current line
smglp	set_left_margin_parm	Zm	Set left margin at column #1 †
smgr	set_right_margin	MR	Set right margin at current column
smgrp	set_right_margin_parm	Zn	Set right margin at column #1 †
smgt	set_top_margin	Zo	Set top margin at current line
smgtp	set_top_margin_parm	Zp	Set top margin at line #1 †
smicm	enter_micro_mode	ZJ	Enable micro motion capabilities
smir	enter_insert_mode	im	Insert mode (enter)
smkx	keypad_xmit	ks	Put terminal in ''keypad-transmit'' mode
smln	label_on	LO	Turn on soft labels
smm	meta_on	mm	Turn on "meta mode" (8th bit)
smso	enter_standout_mode	so	Begin standout mode
smxon	enter_xon_mode	SX	Turn on xon/xoff handshaking
snlq	enter_near_letter_quality	ZK	Set near-letter quality print
snrmq	enter_normal_quality	ZL	Set normal quality print
sshm	enter_shadow_mode	ZM	Enable shadow printing
ssubm	enter_subscript_mode	ZN	Enable subscript printing
ssupm	enter_superscript_mode	ZO	Enable superscript printing
subcs	subscript_characters	Zu	List of ''subscript-able'' characters
sum	enter_upward_mode	ZP	Enable upward carriage motion
supcs	superscript_characters	Zv	List of ''superscript-able'' characters
swidm	enter_doublewide_mode	ZF	Enable double wide printing
tbc	clear_all_tabs	ct	Clear all tab stops
tone	tone	TO	Select touch tone dialing
tsl	to_status_line	ts	Go to status line, col #1 (G)
u0	user0	u0	User string 0
u1	user1	u1	User string 1
u2	user2	u2	User string 2
u3	user3	u3	User string 3
u4	user4	u4	User string 4
u5	user5	u5	User string 5
u6	user6	u6	User string 6
u7	user7	u7	User string 7
u8	user8	u8	User string 8
u9	user9	u9	User string 9

Strings (cont.)

Variable	Cap-name	Termcap Code	Description
uc	underline_char	uc	Underscore one char and move past it
up	cursor_up	cuul	Upline (cursor up)
vpa	row_address	cv	Vertical position absolute (G)
wait	wait_tone	WA	Wait for dial tone
wind	set_window	wi	Current window is lines #1-#2 cols #3-#4 (G)
wingo	goto_window	WG	Got to window #1
wnum	maximum_windows	MW	Maximum number of definable windows
xoffc	xoff_character	XF	X-off character
xonc	xon_character	XN	X-on character
zerom	zero_motion	Zx	No motion for the subsequent character

Sample Entry

The following entry, which describes the AT&T 610 terminal, is among the more complex entries in the *terminfo* file as of this writing.

```
610 | 610bct | ATT610 | att610 | AT&T 610; 80 column; 98key keyboard
        am, eslok, hs, mir, msgr, xenl, xon,
        cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, wsl#80,
        acsc=``aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{ | | }}~~,
        bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
        civis=\E[?25l, clear=\E[H\E[J, cnorm=\E[?25h\E[?12l,
        cr=\r, csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=\b,
        cud=\E[%p1%dB, cud1=\E[B, cuf=\E[%p1%dC, cuf1=\E[C,
        cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
        cvvis=\E[?12;25h, dch=\E[%p1%dP, dch1=\E[P, dim=\E[2m,
        dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K,
        flash=\E[?5h$<200>\E[?5l, fsl=\E8, home=\E[H, ht=\t,
        ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=\ED,
        invis=\E[8m,
        is1=\E[8;0 | \E[?3;4;5;13;15l\E[13;20l\E[?7h\E[12h\E(B\E)0,
        is2=\E[0m^O, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
        kbs=\b, kcbt=\E[Z, kclr=\E[2J, kcub1=\E[D, kcud1=\E[B,
        kcuf1=\E[C, kcuu1=\E[A, kf1=\EOc, kf10=\ENp,
        kf11=\ENq, kf12=\ENr, kf13=\ENs, kf14=\ENt, kf2=\EOd,
        kf3=\EOe, kf4=\EOf, kf5=\EOg, kf6=\EOh, kf7=\EOi,
        kf8=\EOj, kf9=\ENo, khome=\E[H, kind=\E[S, kri=\E[T,
        ll=\E[24H, mc4=\E[?4i, mc5=\E[?5i, nel=\EE,
        pfx=\E[%p1%d;%p2%l%02dq\s\s\sF%p1%1d\s\s\s\s\s
\s\s\s\s\s\s%p2%s,
        pln=\E[%p1%d;0;0;0q%p2%:-16.16s, rc=\E8, rev=\E[7m,
        ri=\EM, rmacs=^O, rmir=\E[4l, rmln=\E[2p, rmso=\E[m,
        rmul=\E[m, rs2=\Ec\E[?3l, sc=\E7,
        sgr=\E[0%?%p6%t;1%;%?%p5%t;2%;%?%p2%t;4%;%?%p4%t;5%;
%?%p3%p1% | %t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
        sgr0=\E[m^O, smacs=^N, smir=\E[4h, smln=\E[p,
        smso=\E[7m, smul=\E[4m, tsl=\E7\E[25;%i%p1%dx,
```

Types of Capabilities in the Sample Entry

The sample entry shows the formats for the three types of *terminfo* capabilities listed: Boolean, Numeric, and String. The names of Boolean capabilities are often listed as abbreviations or acronyms, such as **am** (short for "automatic margins") in the sample entry. ("Automatic margins" is a short description of an automatic return and linefeed when the end of a line is reached.)

Numeric capabilities are followed by the character '#' and then the value. Thus, in the sample, **cols** (which shows the number of columns available on a terminal) gives the value **80** for the AT&T 610. (Values for numeric capabilities may be specified in decimal, octal or hexadecimal, using normal C conventions.)

Finally, string-valued capabilities such as **el** (clear to end of line sequence) are listed by a two- to five-character capname, an '=', and a string ended by the next occurrence of a comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in **$<..>** brackets, as in **el=\EK$<3>**. Padding characters are supplied by **tputs( )**. The delay can be any of the following: a number (**5**), a number followed by a '*' (**5***), a number followed by a '/' (**5/**), or a number followed by both (**5*/**). A '*' shows that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert characters, the factor is still the number of lines affected. This is always 1 unless the terminal has **in** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A '/' indicates that the padding is mandatory. Absence of a '/' is not shown, if the terminal has **xon** defined. Padding information is advisory and will be used only for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both **\E** and **\e** map to an ESCAPE character, *^x* maps to a control-*x* for any appropriate *x*, and the sequences **\n**, **\l**, **\r**, **\t**, **\b**, **\f**, and **\s** give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: \^ for caret (^); \\ for backslash (\); \, for comma (,); \: for colon (:); and **\0** for null. (**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a backslash (e.g., \123).

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *terminfo* and to build up a description gradually, using partial descriptions with *vi*(C) to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **terminfo** file to describe it or the inability of *vi*(C) to work with that terminal. To test a new terminal description, set the environment variable **TERMINFO** to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in */usr/lib/terminfo*. To get the padding for insert-line correct (if the terminal manufacturer did not document it) a severe test is to comment out **xon**, edit a large file at 9600 baud with *vi*(C), delete 16 or so lines from the middle of the screen, then hit the **u** key several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Section 1-1: Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal has a screen, then the number of lines on the screen is given by the **lines** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as the Tektronix 4010 series, as well as hardcopy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (such as a bell or a beep), specify it as **bel**. If the terminal uses the xon-xoff flow-control protocol, like most terminals, specify **xon**.

If there is a code to move the cursor one position to the left (such as backspace), that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over; for example, you would not normally use "**cuf1=\s**" because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and should never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the *terminfo* file usually assumes that this is on; i.e., **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the model 33 teleprinter is described as:

```
hc, os, xon
    cols#72,
    bel=^G, cr=\r, cud1=\n, ind=\n,
```

while the Lear Siegler ADM-3 is described as:

```
adm3 | lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H,
cud1=^J, ind=^J, lines#24,
```

Section 1-2: Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with *printf*(S)-like escapes (%x) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate it in the manner of a Reverse Polish Notation (postfix) calculator. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use **%gx%{5}%-**.

The % encodings have the following meanings:

%%	outputs '%'
%[[:]*flags*][*width*[.*precision*]][**doxXs**]	as in printf, flags are [-+#] and space
%c	print pop() gives %c
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'*c*'	push char constant *c*
%{*nn*}	push decimal constant *nn*
%l	push strlen(pop())
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop() op pop())
%& %| %^	bit operations: push(pop() op pop())
%= %> %<	logical operations: push(pop() op pop())
%A %O	logical operations: and, or
%! %~	unary operations: push(op pop())
%i	(for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present

%? expr %t thenpart %e elsepart %;
if-then-else, %e elsepart is optional;
else-if's are possible ala Algol 68:
%? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e b_5%;
c_i are conditions, b_i are bodies.

If the "-" flag is used with "%[doxXs]", then a colon (:) must be placed between the "%" and the "-" to differentiate the flag from the binary "%-" operator, .e.g., "%:-16.16s".

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its **cup** capability is "**cup**=\E&a%p2%2.2dc%p1%2.2dY$<6>".

The Micro-Term ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, "**cup**=^T%p1%c%p2%c". Terminals which use "%c" need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with *terminfo* set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "**cup=**\E=%p1%'\s'%+%c%p2%'\s'%+%c". After sending "\E=", this pushes the first parameter, pushes the ASCII value for a space (S), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Section 1-3: Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the **\EH** sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

If the terminal has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (e.g., move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the Tektronix 4025.

Section 1-4: Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Section 1-5: Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command -- the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind, ri, indn, rin, dl,** and **dl1** all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Section 1-6: Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using -*terminfo*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the *Concept* 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "**abc def**" using local cursor motions (not spaces) between the **abc** and the **def**. Then position the cursor before the **abc** and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the **abc** shifts over to the **def** which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for "insert null". While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped spaces) no terminals whose insert mode cannot be described with the single attribute have been seen.

terminfo can describe both terminals which have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals which send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will insert *n* blanks.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Section 1-7: Highlighting, Underlining, and Visible Bells

Your terminal may have one or more kinds of display attributes that allow you to highlight selected characters when they appear on the screen. The following display modes (shown with the names by which they are set) may be available: a blinking screen (**blink**), bold or extra-bright characters (**bold**), dim or half-bright characters (**dim**), blanking or invisible text (**invis**), protected text (**prot**), a reverse-video screen (**rev**), and an alternate character set (**smacs** to enter this mode and **rmacs** to exit it). (If a command is necessary before you can enter alternate character set mode, give the sequence in **enacs** or ''enable alternate-character-set'' mode.) Turning on any of these modes singly may or may not turn off other modes.

If you set any display attributes for highlighting, you will also want to provide the capability for turning them off. To do so, set **sgr0**.

You should choose one display method as *standout mode* [see *curses*(S)] and use it to highlight error messages and other kinds of text to which you want to draw attention. Choose a form of display that provides strong contrast but that is easy on the eyes. (We recommend reverse-video plus half-bright or reverse-video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** , respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Micro-Term MIME, this can be given as **uc**.

For historical reasons, some programs interpret **rmso, rmul** to mean "turn off all attributes," not just standout and underline, respectively.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need to be supported by **sgr**; only those for which corresponding separate attribute commands exist should be supported. (See the example at the end of this section.)

Terminals with the ''magic cookie'' glitch (**xmc**) deposit special ''cookies'' when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character.

Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of either of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals, such as the *Concept*, with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the Tektronix 4025, where **smcup** sets the command character to be the one used by **terminfo**. If the **smcup** sequence will not restore the screen after a **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability **ul**. For terminals where a character overstriking another leaves both characters on the screen, give the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Example of highlighting: assume that the terminal under question needs the following escape sequences to turn on various modes.

tparm parameter	attribute	escape sequence
	none	\E[0m
p1	standout	\E[0;4;7m
p2	underline	\E[0;3m
p3	reverse	\E[0;4m

p4	blink	\E[0;5m
p5	dim	\E[0;7m
p6	bold	\E[0;3;4m
p7	invis	\E[0;8m
p8	protect	not available
p9	altcharset	^O (off) ^N(on)

Note that each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, because this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m**. The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N**, depending on whether it is off or on. If all modes were to be turned on, the sequence would be **\E[0;3;4;5;7;8m^N**.

Now look at when different sequences are output. For example, **;3** is output when either **p2** or **p6** is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

sequence	when to output	terminfo translation
\E[0	always	\E[0
;3	if p2 or p6	%?%p2%p6%|%t;3%;
;4	if p1 or p3 or p6	%?%p1%p3%|%p6%|%t;4%;
;5	if p4	%?%p4%t;5%;
;7	if p1 or p5	%?%p1%p5%|%t;7%;
;8	if p7	%?%p7%t;8%;
m	always	m
^N or ^O	if p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the **sgr** sequence gives:

sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p5%t;5%;
%?%p1%p5%|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

Section 1-8: Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to transmit.

The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome** respectively. If there are function keys such as f0, f1, ..., f63, the codes they send can be given as **kf0, kf1, ..., kf63**. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ..., lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1, ka3, kb2, kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. A string to program their soft-screen labels can be given as **pln**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local mode; and **pfx** causes the string to be transmitted to the computer. The capabilities **nlab**, **lw**, and **lh** define how many soft labels there are and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln**. **smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Section 1-9: Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A "backtab" command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by **tput init** [see *tput*(C)] to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the *terminfo* description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the terminal; **iprog**, the path name of a program to be run to initialize the terminal; and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the *terminfo* description. They must be sent to the terminal each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl**, and **smgr**; set the tabs using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of *tput*(C); see *profile*(F).

Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a harder reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from */usr/lib/tabset/**; however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput reset**, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed since the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

Any margin can be cleared with **mgc**. (For instructions on how to specify commands to set and clear margins, see ''Margins'' below under ''PRINTER CAPABILITIES.'')

Section 1-10: Delays

Certain capabilities control padding in the *tty*(7) driver. These are primarily needed by hard-copy terminals, and are used by **tput init** to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Section 1-11: Status Lines

If the terminal has an extra ''status line'' that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, e.g., **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

Section 1-12: Line Graphics

If the terminal has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	vt100+ character
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	-
diamond	`
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (├)	t

right tee (┤)	u
bottom tee (┴)	v
top tee (┬)	w
vertical line	x
bullet	~

The best way to describe a new terminal's line graphics set is to add a third column to the above table with the characters for the new terminal that produce the appropriate glyph when the terminal is in the alternate character set mode. For example,

glyph name	vt100+ character	new tty character
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in "**acsc**=lRmFkTjGq\,x.".

In addition, *terminfo* allows you to define multiple character sets. See Section 2-5 for details.

Section 1-13: Color Manipulation

There are two methods of color manipulation: the HP method and the Tektronix method. Most existing color terminals belong to one of these two classes.

The Tektronix method uses a set of N predefined colors (usually 8) from which a user can select "current" foreground and background colors. Thus the terminal can support up to N colors mixed into N*N color-pairs to be displayed on the screen at the same time.

The HP method restricts the user from defining the foreground independently of the background, or vice-versa. Instead, the user must define an entire color-pair at once. Up to M color-pairs, made from 2*M different colors, can be defined this way.

The numeric variables **colors** and **pairs** define the number of colors and color-pairs that can be displayed on the screen at the same time. If a terminal can change the definition of a color (as can, for example, the Tektronix 4100 and 4200 series terminals), this should be specified with **ccc** (can change color). To change the definition of a color (Tektronix method), use **initc** (initialize color). It requires four arguments: color number (ranging from 0 to **colors**-1) and three RGB (red, green, and blue) values (ranging from 0 to 1,000).

Tektronix 4100 series terminals use a type of color notation called HLS (Hue Lightness Saturation) instead of RGB color notation. For such terminals one must define a boolean variable **hls**. The last three arguments to the **initc** string would then be HLS values: H, ranging from 0 to 360; and L and S, ranging from 0 to 100.

If a terminal can change the definitions of colors, but uses a color notation different from RGB and HLS, a mapping to either RGB or HLS must be developed.

To set current foreground or background to a given color, use **setff** (set foreground) and **setb** (set background). They require one parameter: the number of the color. To initialize a color-pair (HP method), use **initp** (initialize pair). It requires seven parameters: the number of a color-pair (range = 0 to **pairs**-1), and six RGB values: three for the foreground followed by three for the background. (Each of these groups of three should be in the order RGB.) When **initc** or **initp** are used, RGB or HLS arguments should be in the order "red, green, blue" or "hue, lightness, saturation"), respectively. To make a color-pair current, use **scp** (set color-pair). It takes one parameter, the number of a color-pair.

Some terminals (for example, most color terminal emulators for PCs) erase areas of the screen with current background color. In such cases, **bce** (background color erase) should be defined. The variable **op** (original pair) contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, **oc** (original colors) contains a control sequence for setting all colors (for the Tektronix method) or color-pairs (for the HP method) to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. Information about these video attributes should be packed into the **ncv** (no color video) variable. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence.

Attribute	NCV Bit Number
A_STANDOUT	0
A_UNDERLINE	1
A_REVERSE	2
A_BLINK	3
A_DIM	4
A_BOLD	5
A_INVIS	6
A_PROTECT	7

A_ALTCHARSET 8

When a particular video attribute should not be used with colors, the corresponding **ncv** bit should be set to 1; otherwise it should be set to zero. For example, if the terminal uses colors to simulate reverse video and bold, bits 2 and 5 should be set to 1. The resulting values for **ncv** will be 22.

Section 1-14: Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **tparm(repeat_char, 'x', 10)** is the same as **xxxxxxxxxx**.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some UNIX systems: If the environment variable CC exists, all occurrences of the prototype character are replaced with the character in CC.

Terminal descriptions that do not represent a specific kind of known terminal, such as **switch**, **dialup**, **patch**, and **network**, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to **virtual** terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt**. A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi**.

If the terminal uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not ^S and ^Q, they may be specified with **xonc** and **xoffc**.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Section 1-15: Special Cases

The working model used by *terminfo* fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by *terminfo*. These are not to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the *terminfo* model implemented.

Terminals which can not display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals which ignore a linefeed immediately after an **am** wrap, such as the *Concept* 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie" therefore, to erase standout mode, it is instead necessary to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control-C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for control-C.

Section 1-16: Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing *xx@* to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

PART 2: PRINTER CAPABILITIES

The *terminfo* database allows you to define capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under "TERMINAL CAPABILITIES" that list capabilities by variable and by capability name.

Section 2-1: Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that *terminfo* designers create strings that expect numeric values that have been rounded. Application designers should note this and should always round values to the nearest integer before using them with a parameterized string capability.

Section 2-2: Printer Resolution

A printer's resolution is defined to be the smallest spacing of characters it can achieve. In general printers have independent resolution horizontally and vertically. Thus the vertical resolution of a printer can be determined by measuring the smallest achievable distance between consecutive printing baselines, while the horizontal resolution can be determined by measuring the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

All printers are assumed to be capable of printing with a uniform horizontal and vertical resolution. The view of printing that the *terminfo* currently presents is one of printing inside a uniform matrix: All characters are printed at fixed positions relative to each "cell" in the matrix; furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution. (The cell size can be changed as will be seen later.)

Many printers are capable of "proportional printing," where the horizontal spacing depends on the size of the character last printed. The - *terminfo* does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of "moving" to a position an integral multiple of the smallest distance away from a previous position. Thus printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length or width of a single page.

Some printers can have different resolutions depending on different "modes." In "normal mode," the existing *terminfo* capabilities are assumed to work on columns and lines, just like a video terminal. Thus the old **lines** capability would give the length of a page in lines, and the **cols** capability would give the width of a page in columns. In "micro mode," many *terminfo* capabilities work on increments of lines and columns. With some printers the micro mode may be concomitant with normal mode, so that all the capabilities work at the same time.

Section 2-3: Specifying Printer Resolution

The printing resolution of a printer is given in several ways. Each specifies the resolution as the number of smallest steps per distance:

Specification of Printer Resolution

Characteristic	Number of Smallest Steps
orhi	Steps per inch horizontally
orvi	Steps per inch vertically
orc	Steps per column
orl	Steps per line

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution. When printing in micro mode, these distances can be different, and may be zero for some printers.

Specification of Printer Resolution
Automatic Motion after Printing

Normal Mode:	
orc	Steps moved horizontally
orl	Steps moved vertically
Micro Mode:	
mcs	Steps moved horizontally
mls	Steps moved vertically

Some printers are capable of printing wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related: If the distance moved for a regular character is the same whether in normal mode or micro mode (**mcs=orc**), then the distance moved for a wide character is also the same whether in normal mode or micro mode. This doesn't mean the normal character distance is necessarily the same as the wide character distance, just that the distances don't change with a change in normal to micro mode. However, if the distance moved for a regular character is different in micro mode from the distance moved in normal mode (**mcs<orc**), the micro mode distance is assumed to be the same for a wide character printed in micro mode, as the table below shows.

Specification of Printer Resolution
Automatic Motion after Printing Wide Character

Normal Mode or Micro Mode (**mcs = orc**):	
widcs	Steps moved horizontally
Micro Mode (**mcs < orc**):	
mcs	Steps moved horizontally

There may be control sequences to change the number of columns per inch (the character pitch) and to change the number of lines per inch (the line pitch). If these are used, the resolution of the printer changes, but the type of change depends on the printer:

Specification of Printer Resolution
Changing the Character/Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **orhi**, otherwise changes **orc**
lpi	Change line pitch
lpix	If set, **lpi** changes **orvi**, otherwise changes **orl**
chr	Change steps per column

cvr Change steps per line

The **cpi** and **lpi** string capabilities are each used with a single argument, the pitch in columns (or characters) and lines per inch, respectively. The **chr** and **cvr** string capabilities are each used with a single argument, the number of steps per column and line, respectively.

Using any of the control sequences in these strings will imply a change in some of the values of **orc**, **orhi**, **orl**, and **orvi**. Also, the distance moved when a wide character is printed, **widcs**, changes in relation to **orc**. The distance moved when a character is printed in micro mode, **mcs**, changes similarly, with one exception: if the distance is 0 or 1, then no change is assumed (see item marked with † in the following table).

Programs that use **cpi**, **lpi**, **chr**, or **cvr** should recalculate the printer resolution (and should recalculate other values — see "Section 2-7: Effect of Changing Printing Resolution").

Specification of Printer Resolution
Effects of Changing the Character/Line Pitches

Before	*After*
Using **cpi** *with* **cpix** *clear:*	
orhi ’	**orhi**
orc ’	$\mathbf{orc}=\frac{\mathbf{orhi}}{V_{cpi}}$
Using **cpi** *with* **cpix** *set:*	
orhi ’	$\mathbf{orhi}=\mathbf{orc}\cdot V_{cpi}$
orc ’	**orc**
Using **lpi** *with* **lpix** *clear:*	
orvi ’	**orvi**
orl ’	$\mathbf{orl}=\frac{\mathbf{orvi}}{V_{lpi}}$
Using **lpi** *with* **lpix** *set:*	
orvi ’	$\mathbf{orvi}=\mathbf{orl}\cdot V_{lpi}$
orl ’	**orl**
Using **chr:**	
orhi ’	**orhi**
orc ’	V_{chr}
Using **cvr:**	
orvi ’	**orvi**
orl ’	V_{cvr}
Using **cpi** *or* **chr:**	
widcs ’	$\mathbf{widcs}=\mathbf{widcs}'\frac{\mathbf{orc}}{\mathbf{orc}'}$
mcs ’ †	$\mathbf{mcs}=\mathbf{mcs}'\frac{\mathbf{orc}}{\mathbf{orc}'}$

V_{cpi}, V_{lpi}, V_{chr}, and V_{cvr} are the arguments used with **cpi, lpi, chr,** and **cvr** respectively. The † mark indicates the old value.

Section 2-4: Capabilities that Cause Movement

In the following descriptions, ''movement'' refers to the motion of the ''current position.'' With video terminals this would be the cursor; with some printers this is the carriage position. Other printers have different equivalents. In general, the current position is where a character would be displayed if printed.

terminfo has string capabilities for control sequences that cause movement a number of full columns or lines. It also has equivalent string capabilities for control sequences that cause movement a number of smallest steps.

String Capabilities for Motion

mcub1	Move 1 step left
mcuf1	Move 1 step right
mcuu1	Move 1 step up
mcud1	Move 1 step down
mcub	Move *N* steps left
mcuf	Move *N* steps right
mcuu	Move *N* steps up
mcud	Move *N* steps down
mhpa	Move *N* steps from the left
mvpa	Move *N* steps from the top

The latter six strings are each used with a single argument, *N*.

Sometimes the motion is limited to less than the width or length of a page. Also, some printers don't accept absolute motion to the left of the current position. *terminfo* has capabilities for specifying these limits.

Limits to Motion

mjump	Limit on use of **mcub1, mcuf1, mcuu1, mcud1**
maddr	Limit on use of **mhpa, mvpa**
xhpa	If set, **hpa** and **mhpa** can't move left
xvpa	If set, **vpa** and **mvpa** can't move up

If a printer needs to be in a ''micro mode'' for the motion capabilities described above to work, there are string capabilities defined to contain the control sequence to enter and exit this mode. A boolean is available for those printers where using a carriage return causes an automatic return to normal mode.

Entering/Exiting Micro Mode	
smicm	Enter micro mode
rmicm	Exit micro mode
crxm	Using **cr** exits micro mode

The movement made when a character is printed in the rightmost position varies among printers. Some make no movement, some move to the beginning of the next line, others move to the beginning of the same line. *terminfo* has boolean capabilities for describing all three cases.

What Happens After Character Printed in Rightmost Position	
sam	Automatic move to beginning of same line

Some printers can be put in a mode where the normal direction of motion is reversed. This mode can be especially useful when there exists no capabilities for leftward or upward motion, because those capabilities can be built from the motion reversal capability and the rightward or downward motion capabilities. It is best to leave it up to an application to build the leftward or upward capabilities, though, and not enter them in the *terminfo* database. This allows several reverse motions to be strung together without intervening wasted steps that leave and reenter reverse mode.

Entering/Exiting Reverse Modes	
slm	Reverse sense of horizontal motions
rlm	Restore sense of horizontal motions
sum	Reverse sense of vertical motions
rum	Restore sense of vertical motions
While sense of horizontal motions reversed:	
mcub1	Move 1 step right
mcuf1	Move 1 step left
mcub	Move *N* steps right
mcuf	Move *N* steps left
cub1	Move 1 column right
cuf1	Move 1 column left
cub	Move *N* columns right
cuf	Move *N* columns left
While sense of vertical motions reversed:	
mcuu1	Move 1 step down
mcud1	Move 1 step up
mcuu	Move *N* steps down
mcud	Move *N* steps up
cuu1	Move 1 line down
cud1	Move 1 line up

cuu	Move *N* lines down
cud	Move *N* lines up

The reverse motion modes should not affect the **mvpa** and **mhpa** absolute motion capabilities. The reverse vertical motion mode should, however, also reverse the action of the line "wrapping" that occurs when a character is printed in the right-most position. Thus printers that have the standard *terminfo* capability **am** defined should experience motion to the beginning of the previous line when a character is printed in the right-most position under reverse vertical motion mode.

The action when any other motion capabilities are used in reverse motion modes is not defined; thus, programs must exit reverse motion modes before using other motion capabilities.

Two miscellaneous capabilities complete the list of new motion capabilities. One of these is needed for printers that move the current position to the beginning of a line when certain control characters, like "line-feed" or "form-feed," are used. The other is used for the capability of suspending the motion that normally occurs after printing a character.

Miscellaneous Motion Strings	
docr	List of control characters causing **cr**
zerom	Prevent auto motion after printing next single character

Margins

terminfo provides two strings for setting margins on terminals: one for the left and one for the right margin. Printers, however, have two additional margins, for the top and bottom margins of each page. Furthermore, some printers do not require using motion strings to move the current position to a margin and fixing the margin there, as with the existing capabilities, but require the specification of where a margin should be regardless of the current position. Therefore *terminfo* offers six additional strings for defining margins with printers.

Setting Margins	
smgl	Set left margin at current column
smgr	Set right margin at current column
smgb	Set soft bottom margin at current line
smgt	Set soft top margin at current line
smgbp	Set soft bottom margin at line *N*
smglp	Set soft left margin at column *N*

smgrp	Set soft right margin at column *N*
smgtp	Set soft top margin at line *N*

The last four strings are used with a single argument, *N*, that gives the line or column number, where line 0 is the top line and column 0 is the left-most column. Note: Not all printers use 0 for the top line or the left-most column.

All margins can be cleared with **mgc**.

Shadows, Italics, Wide Characters, Superscripts, Subscripts

Five new sets of strings are used to describe the capabilities printers have of enhancing printed text.

Enhanced Printing	
sshm	Enter shadow-printing mode
rshm	Exit shadow-printing mode
sitm	Enter italicizing mode
ritm	Exit italicizing mode
swidm	Enter wide character mode
rwidm	Exit wide character mode
ssupm	Enter superscript mode
rsupm	Exit superscript mode
supcs	List of characters available as superscripts
ssubm	Enter subscript mode
rsubm	Exit subscript mode
subcs	List of characters available as subscripts

If a printer requires the **sshm** control sequence before every character to be shadow-printed, the **rshm** string is left blank. Thus programs that find a control sequence in **sshm** but none in **rshm** should use the **sshm** control sequence before every character to be shadow-printed; otherwise, the **sshm** control sequence should be used once before the set of characters to be shadow-printed, followed by **rshm**. The same is also true of each of the **sitm/ritm**, **swidm/rwidm**, **ssupm/rsupm**, and **ssubm/rsubm** pairs.

Note that *terminfo* also has a capability for printing emboldened text (**bold**). While shadow printing and emboldened printing are similar in that they "darken" the text, many printers produce these two types of print in slightly different ways. Generally, emboldened printing is done by overstriking the same character one or more times. Shadow printing likewise usually involves overstriking, but with a slight movement up and/or to the side so that the character is "fatter."

It is assumed that enhanced printing modes are independent modes, so that it would be possible, for instance, to shadow print italicized subscripts.

As mentioned earlier, the amount of motion automatically made after printing a wide character should be given in **widcs**.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in **supcs** or **subcs** strings, respectively. If the **ssupm** or **ssubm** strings contain control sequences, but the corresponding **supcs** or **subcs** strings are empty, it is assumed that all printable ASCII characters are available as superscripts or subscripts.

Automatic motion made after printing a superscript or subscript is assumed to be the same as for regular characters. Thus, for example, printing any of the following three examples will result in equivalent motion:

$$\mathbf{Bi}\ \mathbf{B}_{\mathrm{i}}\ \mathbf{B}^{\mathrm{i}}$$

Note that the existing **msgr** boolean capability describes whether motion control sequences can be used while in "standout mode." This capability is extended to cover the enhanced printing modes added here. **msgr** should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if **msgr** is not set, a program should end these modes before attempting any motion.

Section 2-5: Alternate Character Sets

In addition to allowing you to define line graphics (described in Section 1-12), *terminfo* also lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets.

Alternate Character Sets

scs	Select character set *N*
scsd	Start definition of character set *N*, *M* characters
defc	Define character *A*, *B* dots wide, descender *D*
rcsd	End definition of character set *N*
csnm	List of character set names
daisy	Printer has manually changed print-wheels

The **scs**, **rcsd**, and **csnm** strings are used with a single argument, *N*, a number from 0 to 63 that identifies the character set. The **scsd** string is also used with the argument *N* and another, *M*, that gives the number of characters in the set. The **defc** string is used with three arguments: *A* gives the ASCII code representation for the character, *B* gives the width of the character in dots, and *D* is zero or one depending on whether the character is a "descender" or not. The **defc** string is also followed by a string of "image-data" bytes that describe how the character looks (see below).

Character set 0 is the default character set present after the printer has been initialized. Not every printer has 64 character sets, of course; using **scs** with an argument that doesn't select an available character set should cause a null result from *tparm*( ).

If a character set has to be defined before it can be used, the **scsd** control sequence is to be used before defining the character set, and the **rcsd** is to be used after. They should also cause a null result from *tparm*( ) when used with an argument *N* that doesn't apply. If a character set still has to be selected after being defined, the **scs** control sequence should follow the **rcsd** control sequence. By examining the results of using each of the **scs**, **scsd**, and **rcsd** strings with a character set number in a call to *tparm*( ), a program can determine which of the three are needed.

Between use of the **scsd** and **rcsd** strings, the **defc** string should be used to define each character. To print any character on printers covered by *terminfo*, the ASCII code is sent to the printer. This is true for characters in an alternate set as well as ''normal'' characters. Thus the definition of a character includes the ASCII code that represents it. In addition, the width of the character in dots is given, along with an indication of whether the character should descend below the print line (like the lower case letter ''g'' in most character sets). The width of the character in dots also indicates the number of image-data bytes that will follow the **defc** string. These image-data bytes indicate where in a dot-matrix pattern ink should be applied to ''draw'' the character; the number of these bytes and their form are defined below under "Dot-Mapped Graphics".

It's easiest for the creator of *terminfo* entries to refer to each character set by number; however, these numbers will be meaningless to the application developer. The **csnm** string alleviates this problem by providing names for each number.

When used with a character set number in a call to *tparm*( ), the **csnm** string will produce the equivalent name. These names should be used as a reference only. No naming convention is implied, although anyone who creates a *terminfo* entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the **csnm** string to determine the correct number), or by name, where the application examines the **csnm** string to determine the corresponding character set number.

These capabilities are likely to be used only with dot-matrix printers. If they are not available, the strings should not be defined. For printers that have manually changed print-wheels or font cartridges, the boolean **daisy** is set.

Section 2-6: Dot-Matrix Graphics

Dot-matrix printers typically have the capability of reproducing "raster-graphics" images. Three new numeric capabilities and three new string capabilities can help a program draw raster-graphics images independent of the type of dot-matrix printer or the number of pins or dots the printer can handle at one time.

Dot-Matrix Graphics

npins	Number of pins, *N*, in print-head
spinv	Spacing of pins vertically in pins per inch
spinh	Spacing of dots horizontally in dots per inch
porder	Matches software bits to print-head pins
sbim	Start printing bit image graphics, *B* bits wide
rbim	End printing bit image graphics

The **sbim** sring is used with a single argument, *B*, the width of the image in dots.

The model of dot-matrix or raster-graphics that the *terminfo* presents is similar to the technique used for most dot-matrix printers: Each pass of the printer's print-head is assumed to produce a dot-matrix that is *N* dots high and *B* dots wide. This is typically a wide, squat, rectangle of dots. The height of this rectangle in dots will vary from one printer to the next; this is given in the **npins** numeric capability. The size of the rectangle in fractions of an inch will also vary; it can be deduced from the **spinv** and **spinh** numeric capabilities. With these three values an application can divide a complete raster-graphics image into several horizontal strips, perhaps interpolating to account for different dot spacing vertically and horizontally.

The **sbim** and **rbim** strings are used to start and end a dot-matrix image, respectively. The **sbim** string is used with a single argument that gives the width of the dot-matrix in dots. A sequence of "image-data bytes" are sent to the printer after the **sbim** string and before the **rbim** string. The number of bytes is an integral multiple of the width of the dot-matrix; the multiple and the form of each byte is determined by the **porder** string as described below.

The **porder** string is a comma separated list of pin numbers; the position of each pin number in the list corresponds to a bit in a data byte. The pins are numbered consecutively from 1 to **npins**, with 1 being the top pin. Note that the term "pin" is used loosely here; "ink-jet" dot-matrix printers don't have pins, but can be considered to have an equivalent method of applying a single dot of ink to paper. The bit positions in **porder** are in groups of 8, with the first position in each group the most significant bit and the last position the least significant bit.

The "image-data bytes" are to be computed from the dot-matrix image, mapping vertical dot positions in each print-head pass into eight-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. If a position is skipped in **porder**, a 0 bit is used. There must be a multiple of 8 bit positions used or skipped in **porder**; if not, 0 bits are used to fill the last byte in the least significant bits.

Section 2-7: Effect of Changing Printing Resolution

If the control sequences to change the character pitch or the line pitch are used, the pin or dot spacing may change:

Dot-Matrix Graphics
Changing the Character/Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **spinh**
lpi	Change line pitch
lpix	If set, **lpi** changes **spinv**

Programs that use **cpi** or **lpi** should recalculate the dot spacing:

Dot-Matrix Graphics
Effects of Changing the Character/Line Pitches

Before	*After*
Using **cpi** *with* **cpix** *clear:*	
spinh ’	**spinh**
Using **cpi** *with* **cpix** *set:*	
spinh ’	$\mathbf{spinh}=\mathbf{spinh}'\cdot\frac{\mathbf{orhi}}{\mathbf{orhi}'}$
Using **lpi** *with* **lpix** *clear:*	
spinv ’	**spinv**
Using **lpi** *with* **lpix** *set:*	
spinv ’	$\mathbf{spinv}=\mathbf{spinv}'\cdot\frac{\mathbf{orhi}}{\mathbf{orhi}'}$
Using **chr:**	
spinh ’	**spinh**
Using **cvr:**	
spinv ’	**spinv**

orhi’ and **orhi** are the values of the horizontal resolution in steps per inch, before using **cpi** and after using **cpi**, respectively. Likewise, **orvi’** and **orvi** are the values of the vertical resolution in steps per inch, before using **lpi** and after using **lpi**, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.

Section 2-8: Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near "letter quality" printing or "draft quality" printing. Usually it is important to be able to choose one or the other because the rate of printing generally falls off as the quality improves. There are three new strings used to describe these capabilities.

Print Quality	
snlq	Set near-letter quality print
snrmq	Set normal quality print
sdrfq	Set draft quality print

The capabilities are listed in decreasing levels of quality. If a printer doesn't have all three levels, one or two of the strings should be left blank as appropriate.

Section 2-9: Printing Rate and Buffer Size

Because there is no standard protocol that can be used to keep a program synchronized with a printer, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has been printed. Two new numeric capabilities can help a program estimate what has been printed.

Print Rate/Buffer Size	
cps	Nominal print rate in characters per second
bufsz	Buffer capacity in characters

cps is the nominal or average rate at which the printer prints characters; if this value is not given, the rate should be estimated at one-tenth the prevailing baud rate. **bufsz** is the maximum number of subsequent characters buffered before the guaranteed printing of an earlier character, assuming proper flow control has been used. If this value is not given it is assumed that the printer does not buffer characters, but prints them as they are received.

As an example, if a printer has a 1000-character buffer, then sending the letter "a" followed by 1000 additional characters is guaranteed to cause the letter "a" to print. If the same printer prints at the rate of 100 characters per second, then it should take 10 seconds to print all the characters in the buffer, less if the buffer is not full. By keeping track of the characters sent to a printer, and knowing the print rate and buffer size, a program can synchronize itself with the printer.

Note that most printer manufacturers advertise the maximum print rate, not the nominal print rate. A good way to get a value to put in for **cps** is to generate a few pages of text, count the number of printable characters, then see how long it takes to print the text.

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in **cps**. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in **cps**. If the application is using **cps** to decide how long it should take a printer to print a block of text, the application should pad the estimate. If the application is using **cps** to decide how much text has already been printed, it should shrink the estimate. The application will thus err in favor of the user, who wants, above all, to see all the output in its correct place.

Files

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.COREterm/?/*	subset of compiled terminal description database
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

See Also

tput(C), vi(C), captoinfo(ADM), infocmp(ADM), tic(C), term(M), curses(S), printf(S), profile(F), terminfo(F)

Warning

As described in the "Tabs and Initialization" section above, a terminal's initialization strings, **is1**, **is2**, and **is3**, if defined, must be output before a *curses*(S) program is run. An available mechanism for outputting such strings is **tput init** [see *tput*(C) and *profile*(F)].

If a null character (\0) is encountered in a string, the null and all characters after it are lost. Therefore it is not possible to code a null character (\0) and send it to a device (either terminal or printer). The suggestion of sending a \0200, where a \0 (null) is needed can succeed only if the device (terminal or printer) ignores the eighth bit. For example, because all eight bits are used in the standard international ASCII character set, devices that adhere to this standard will treat \0200 differently from \0.

Tampering with entries in */usr/lib/.COREterm/?/** or */usr/lib/terminfo/?/** (for example, changing or removing an entry) can affect programs such as *vi*(C) that expect the entry to be present and correct. In particular, removing the description for the "dumb" terminal will cause unexpected problems.

termio

general terminal interface

Description

All asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by *getty*(M) and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the "control terminal" for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(S). A process can break this association by changing its process group using *setpgrp*(S).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters can be entered at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been entered. Also, no matter how many characters are requested in the read call, one line will be returned at most. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

Erase and kill processing is normally done during input. By default, a Ctrl-H or BACKSPACE erases the last character typed, except that it will not erase beyond the beginning of the line. By default, a Ctrl-U kills (deletes) the entire input line, and optionally outputs a newline character. Both these characters operate on a key-stroke basis, independent of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case, the escape character is not read. The erase and kill characters may be changed (see *stty*(C)).

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) Generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(S).

QUIT (Ctrl-\ or ASCII FS) Generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated, but a core image file (called core) will be created in the current working directory.

SWTCH (ASCII NUL) Is used by the job control facility, *shl*(C), to change the current layer to the control layer.

ERASE (Ctrl-H) Erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (Ctrl-U) Deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Ctrl-D or ASCII EOT) May be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF) Is the normal line delimiter. It cannot be changed or escaped.

EOL (ASCII NUL) Is an additional line delimiter, like NL. It is not normally used.

STOP (Ctrl-S or ASCII DC3) Temporarily suspends output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Ctrl-Q or ASCII DC1) Resumes output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The START/STOP characters cannot be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding backslash (\) character, in which case no special function is carried out.

When the carrier signal from the dataset drops, a "hangup" signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for an end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as the previously typed characters have been entered. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds a given limit. When the queue has drained down to the given threshold, the program is resumed.

Several *ioctl*(S) system calls apply to terminal files. The primary calls use the following structure, defined in the file **<termio.h>**:

```
#define NCC      8
struct  termio {
        unsigned short  c_iflag;    /* input modes */
        unsigned short  c_oflag;    /* output modes */
        unsigned short  c_cflag;    /* control modes */
        unsigned short  c_lflag;    /* local modes */
        char            c_line;     /* line discipline */
        unsigned char   c_cc[NCC];/* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	Ctrl-H
3	VKILL	Ctrl-U
4	VEOF/VMIN	EOT
5	VEOL/VTIME	NUL
6	VEOL2	EOL
7	VSWTCH	NUL

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignores break condition
BRKINT	0000002	Signals interrupt on break
IGNPAR	0000004	Ignores characters with parity errors
PARMRK	0000010	Marks parity errors
INPCK	0000020	Enables input parity check
ISTRIP	0000040	Strips character
INLCR	0000100	Maps NL to CR on input
IGNCR	0000200	Ignores CR

ICRNL	0000400	Maps CR to NL on input
IUCLC	0001000	Maps uppercase to lowercase on input
IXON	0002000	Enables start/stop output control
IXANY	0004000	Enables any character to restart output
IXOFF	0010000	Enables start/stop input control

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise, if BRKINT is set the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the 3-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character will restart output which has been suspended.

If IXOFF is set, the system will transmit START characters when the input queue is nearly empty and STOP characters when nearly full.

If CTSFLOW or RTSFLOW are set, IXON and IXANY should also be set so that these two types of flow control do not interfere with each other.

The RTS and CTS lines for the RS-232 (i.e. serial) interface were originally intended as handshaking signals between a Data Terminal Equipment (DTE) device (computer, printer, etc,) and a Data Communications Equipment (DCE) device (almost always a modem).

The RTS (Ready To Send) line is asserted by the DTE when it is ready to send data to the DCE. The DCE asserts the CTS (Clear To Send) line when it was ready to receive data. If the CTS line goes low, then the DTE should stop sending data until CTS goes high again.

UNIX systems also use the RTS line for handshaking in the other direction. If the system sees that its input buffer is nearly full, it will lower the RTS line. The serial device should then stop sending, and wait for the system to catch up. The system will raise the RTS line when it is ready for more data.

The initial input control value is all bits clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocesses output
OLCUC	0000002	Maps lowercase to uppercase on output
ONLCR	0000004	Maps NL to CR-NL on output
OCRNL	0000010	Maps CR to NL on output
ONOCR	0000020	No CR output at column 0
ONLRET	0000040	NL performs CR function
OFILL	0000100	Uses fill characters for delay
OFDEL	0000200	Fills is DEL, else NUL
NLDLY	0000400	Selects newline delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Selects carriage return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Selects horizontal tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expands tabs to spaces
BSDLY	0020000	Selects backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Selects vertical tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Selects form feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to perform the carriage return function and the column pointer is set to 0 and the delays specified for CR will be used. Otherwise, the NL character is assumed to perform the linefeed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form feed or vertical tab delay is specified, it lasts for about 2 seconds.

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage return delays are used instead of the newline delays. If OFILL is set, 2 fill characters will be transmitted.

Carriage return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits 2 fill characters, and type 2 transmits 4 fill characters.

Horizontal tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, 2 fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, 1 fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud

B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Sends two stop bits, else one
CREAD	0000200	Enables receiver
PARENB	0000400	Parity enable
PARODD	0001000	Odd parity, else even
HUPCL	0002000	Hangs up on last close
CLOCAL	0004000	Local line, else dial-up
LOBLK	0010000	Block layer output
CTSFLOW	0020000	Enables CTS protocol for a modem line
RTSFLOW	0040000	Enables RTS signaling for a modem line

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Without this signal, the line is disconnected if it is connected through a modem. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, 2 stop bits are used, otherwise 1 stop bit. For example, at 110 baud, 2 stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. The data-terminal-ready and request-to-send signals are asserted, but incoming modem signals are ignored. If CLOCAL is not set, modem control is assumed. This means the data-terminal-ready and request-to-send signals are asserted. Also, the carrier-detect signal must be returned before communications can proceed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B9600, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

```
ISIG      0000001  Enable signals
ICANON    0000002  Canonical input (erase and kill processing)
XCASE     0000004  Canonical upper/lower presentation
ECHO      0000010  Enables echo
ECHOE     0000020  Echoes erase character as BS-SP-BS
ECHOK     0000040  Echoes NL after kill character
ECHONL    0000100  Echoes NL
NOFLSH    0000200  Disables flush after interrupt or quit
XCLUDE    0100000 Exclusive use of the line
```

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus, these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least VMIN characters have been received or the timeout value VTIME has expired and at least one character has been input. This allows fast bursts of input to be read efficiently while still allowing single character input. (See the discussion of VMIN and VTIME below.)

The VMIN and VTIME values are stored in the position for the EOF and EOL characters respectively. VMIN and VTIME are interpreted as EOF and EOL if ICANON is set. Default VMIN and VTIME values are stored in the **/usr/include/sys/termio.h** file. To change these values, set ICANON to off and use *stty*(C) to change the VMIN and VTIME values as represented by EOF and EOL. The TIME value represents tenths of seconds.

If XCASE and ICANON are set, an uppercase letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

<table>
<tr><td>For:</td><td>Use:</td></tr>
<tr><td>`</td><td>\'</td></tr>
<tr><td>|</td><td>\!</td></tr>
<tr><td>~</td><td>\^</td></tr>
<tr><td>{</td><td>\(</td></tr>
<tr><td>}</td><td>\)</td></tr>
<tr><td>\</td><td>\\</td></tr>
</table>

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

If XCLUDE is set, any subsequent attempt to open the TTY device using *open*(S) will fail for all users except the super-user. If the call fails, it returns EBUSY in *errno*. XCLUDE is useful for programs which must have exclusive use of a communications line. It is not intended for the line to the program's controlling terminal. XCLUDE must be cleared before the setting program terminates, otherwise subsequent attempts to open the device will fail.

VMIN represents the minimum number of characters that should be received when the read is satisfied (i.e., the characters are returned to the user). VTIME is a timer of 0.10 second granularity used to time-out bursty and short-term data transmissions. The four possible values for VMIN and VTIME and their interactions are:

VMIN > 0, VTIME > 0
In this case, VTIME serves as an inter-character timer activated after the first character is received, and reset upon receipt of each character. VMIN and VTIME interact as follows:

> As soon as one character is received the inter-character timer is started.

If VMIN characters are received before the inter-character timer expires the read is satisfied.

If the timer expires before VMIN characters are received the characters received to that point are returned to the user.

A *read*(S) operation will sleep until the VMIN and VTIME mechanisms are activated by the receipt of the first character; thus, at least one character must be returned.

VMIN > 0, VTIME = 0
In this case, because VTIME = 0, the timer plays no role and only VMIN is significant. A *read*(S) operation is not satisfied until VMIN characters are received.

VMIN = 0, VTIME > 0
In this case, because VMIN = 0, VTIME no longer serves as an inter-character timer, but now serves as a read timer that is activated as soon as the *read*(S) operation is processed. A *read*(S) operation is satisfied as soon as a single character is received or the timer expires, in which case, the *read*(S) operation will not return any characters.

VMIN = 0, VTIME = 0
In this case, return is immediate. If characters are present, they will be returned to the user.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(S) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA — Gets the parameters associated with the terminal and stores them in the *termio* structure referenced by **arg**.

TCSETA — Sets the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW — Waits for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF — Waits for the output to drain, then flushes the input queue and sets the new parameters.

Additional *ioctl*(S) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK Waits for the output to drain. If *arg* is 0, then sends a break (zero bits for 0.25 seconds).

TCXONC Starts/stops control. If *arg* is 0, suspends output; if 1, restarts suspended output.

TCFLSH If *arg* is 0, flushes the input queue; if 1, flushes the output queue; if 2, flushes both the input and output queues.

Files

/dev/tty

/dev/tty*

/dev/console

See Also

fork(S), ioctl(S), mapchan(F), mapchan(M), read(S), setgprp(S), signal(S), stty(C), tty(M), termios(M)

Standards Conformance

termio is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

termios

POSIX general terminal interface

Description

This page discusses the POSIX *termios* extensions to the *termio*(M) interface. Only those functions not described in *termio*(M) are described here.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

SUSP (ASCII NUL) If the ISIG flag is enabled, receipt of the SUSP character causes A SIGTSTP signal to be sent to the current process group. The SUSP character is discarded when processed. It is normally set to Ctrl-Z.

Several library functions apply to terminal files. The primary calls use the following structure, defined in the file **<termios.h>**:

```
#define NCCS    13
struct termios {
    tcflag_t    c_iflag;     /* input modes */
    tcflag_t    c_oflag;     /* output modes */
    tcflag_t    c_cflag;     /* control modes */
    tcflag_t    c_lflag;     /* local (line discipline) modes */
    char        c_line;      /* line discipline */
    cc_t        c_cc[NCCS];  /* control chars */
    char        c_ispeed;    /* input baud rate */
    char        c_ospeed;    /* output baud rate */
};
```

The additional special control characters defined by the array *c_cc*. are:

```
10    VSUSP  NUL
11    VSTART DC1
12    VSTOP  DC3
```

The following additional line discipline (0) functions are available in the *c_lflag* field:

```
IEXTEN    0000400  enable extended functions
TOSTOP    0001000  SIGTTOU on background output
```

If IEXTEN is set, additional non-POSIX functions are recognized. This is the default. If IEXTEN is not set, the modes ICANON, ISIG, IXON, and IXOFF are assumed.

If TOSTOP is set, the signal SIGTTOU is sent to the process group of a process that tries to write to its controlling terminal if it is not the foreground process group. By default, this signal stops the members of the process group. If TOSTOP is not set, the output generated by the process is output to the current output stream.

The associated library functions are found in *tcattr*(S) and *tcflow*(S).

Files

/dev/tty

/dev/tty*

/dev/console

See Also

ioctl(S), signal(S), stty(C), tcattr(S), tcflow(S), termio(M), tty(M)

Standards Conformance

termios is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and The X/Open Portability Guide II of January 1987.

timod

Transport Interface cooperating STREAMS module

Description

timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library. The *timod* module converts a set of *ioctl*(S) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The *timod* module must only be pushed (see *Streams Primer*) onto a *stream* terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the *ioctl* commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following *ioctl* commands are recognized and processed by the *timod* module. The format of the *ioctl* call is:

```
#include <sys/stropts.h>
                    -
                    -
struct strioctl strioctl;
                    -
                    -
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf

ioctl(fildes, I_STR, &strioctl);
```

where, on issuance, *size* is the size of the appropriate TI message to be sent to the transport provider and on return, *size* is the size of the appropriate TI message from the transport provider in response to the issued TI message. *buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in **<sys/tihdr.h>**. The possible values for the *cmd* field are:

TI_BIND — Bind an address to the underlying transport protocol provider. The message issued to the TI_BIND *ioctl* is equivalent to the TI message type T_BIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_BIND_ACK.

TI_UNBIND Unbind an address from the underlying transport protocol provider. The message issued to the TI_UNBIND *ioctl* is equivalent to the TI message type T_UNBIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OK_ACK.

TI_GETINFO Get the TI protocol specific information from the transport protocol provider. The message issued to the TI_GETINFO *ioctl* is equivalent to the TI message type T_INFO_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_INFO_ACK.

TI_OPTMGMT Get, set, or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT *ioctl* is equivalent to the TI message type T_OPTMGMT_REQ, and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OPTMGMT_ACK.

Files

<sys/timod.h>
<sys/tiuser.h>
<sys/tihdr.h>
<sys/errno.h>

See Also

tirdwr(M)
STREAMS Primer
STREAMS Programmer's Guide
Network Programmer's Guide

Diagnostics

If the *ioctl* system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in **<sys/tiuser.h>**. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in **<sys/errno.h>** [see *intro*(S)].

timtbl

create a time locale table

Syntax

timtbl [*specfile*]

Description

The utility *timtbl* is provided to allow new LC_TIME locales to be defined. It reads a specification file, which contains definitions of the way in which time and date information is presented for a particular locale, and produces a binary table file, to be read by *setlocale* (S), which determines the behavior of the *strftime* (S) routine.

The information supplied in the specification file consists of lines in the following format:

item = *string*

The "=" can be separated from the item and string fields by zero or more space or tab characters. The following values are meaningful for *item*:

DATE_FMT	specification of the format string for representing the date. It will contain "%" directives representing variable items such as the month number, as used in the format string for *strftime* (S).
TIME_FMT	specification of the format string for representing the time of day.
F_NOON	string indicating 12-hour clock times before midday, e.g. "AM".
A_NOON	string indicating 12-hour clock times after midday, e.g. "PM".
D_T_FMT	string for formatting combined date and time.
DAY_1	full name of the first day of the week (Sunday).

.
.
.

DAY_7	full name of the seventh day of the week.
ABDAY_1	abbreviated name of the first day of the week, e.g. ''Sun''.
	. . .
ABDAY_7	abbreviated name of the seventh day of the week.
MON_1	full name of the first month in the Gregorian calendar.
	. . .
MON_12	full name of the twelfth month.
ABMON_1	abbreviated name of the first month.
	. . .
ABMON_12	full name of the twelfth month.

The *string* is a sequence of characters surrounded by quotes ("). Characters within the string can be specified both literally and using ''\'' escapes; the following three strings are equivalent:

"Tuesday"	- literal
"\x54ue\x73da\x79"	- hexadecimal escapes
"\124ue\163da\171"	- octal escapes

The *string*s for the *item*s DATE_FMT , TIME_FMT and D_T_FMT will also include ''%'' directives as detailed in the *strftime*(S) manual page, to specify variable portions of the string.

All characters following a hash (''#'') are treated as a comment and ignored up to the end of the line, unless the hash is within a quoted string.

The various *item*s may be specified in any order. If any items are not specified, a warning message will be produced, and the null string ("") substituted.

The binary table output is placed in a file named ''time'', within the current directory. This file should be copied or linked to the correct place in the *setlocale* file tree (see *locale*(M)). To prevent accidental corruption of the output data, the file is created with no write permission; if the timtbl utility is run in a directory containing a write-protected ''ctype'' file, the utility will ask if the existing file should be replaced - any response other than ''yes'' or ''y'' will cause **timtbl** to terminate without overwriting the existing file.

If the *specfile* argument is missing, the specification information is read from the standard input.

See Also

chrtbl(M), locale(M), numtbl(M), setlocale(S), strftime(S)

Diagnostics

If the input table file cannot be opened for reading, processing will terminate with the error message, "Cannot open specification file".

Any lines in the specification file which are syntactically incorrect, or contain an urecognized value for the *item*, will cause an error message to be issued to the standard error output, specifying the line number on which the error was detected. The line will be ignored, and processing will continue.

If a particular *item* is specified more than once, a warning message will be produced, and processing will continue.

If the specification file does not contain specifications for all possible *item*s, a warning message will be produced.

If the output file, *time* , cannot be opened for writing, processing will terminate with the error message, "Cannot create table file".

Any error conditions encountered will cause the program to exit with a non-zero return code; successful completion is indicated with a zero return code.

Notes

The strings D_FMT , T_FMT , AM_STR and PM_STR may be used as alternatives to DATE_FMT , TIME_FMT , F_NOON and A_NOON respectively, if required. These alternatives are provided for consistency with the identifiers used by *nl_langinfo*(S).

Value Added

timtbl is an extension of AT&T System V provided in Altos UNIX System V.

tirdwr

Transport Interface read/write interface STREAMS module

Description

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the *read*(S) and *write*(S) system calls. The *putmsg*(S) and *getmsg*(S) system calls may also be used. However, *putmsg* and *getmsg* can only transfer data messages between user and *stream.*

The *tirdwr* module must only be pushed [see I_PUSH in *streamio*(M)] onto a *stream* terminated by a transport protocol provider which supports the TI. After the *tirdwr* module has been pushed onto a *stream,* none of the Transport Interface functions can be used. Subsequent calls to TI functions will cause an error on the *stream.* Once the error is detected, subsequent system calls on the *stream* will return an error with *errno* set to EPROTO.

The following are the actions taken by the *tirdwr* module when pushed on the *stream,* popped [see I_POP in *streamio*(M)] off the *stream,* or when data passes through it.

push - When the module is pushed onto a *stream,* it will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the *stream* that relate to process management, such as messages that generate signals to the user processes associated with the *stream.* If any other messages are present, the I_PUSH will return an error with *errno* set to EPROTO.

write - The module will take the following actions on data that originated from a *write* system call:

- All messages with the exception of messages that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's downstream neighbor.

- Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.

- Any messages with control portions will generate an error, and any further system calls associated with the *stream* will fail with *errno* set to EPROTO.

read - The module will take the following actions on data that originated from the transport protocol provider:

- All messages with the exception of those that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's upstream neighbor.
- The action taken on messages with control portions will be as follows:
    + Messages that represent expedited data will generate an error. All further system calls associated with the *stream* will fail with *errno* set to EPROTO.
    + Any data messages with control portions will have the control portions removed from the message prior to passing the message to the upstream neighbor.
    + Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the *stream*. The orderly release message itself will be freed by the module.
    + Messages that represent an abortive disconnect indication from the transport provider will cause all further *write* and *putmsg* system calls to fail with *errno* set to ENXIO. All further *read* and *getmsg* system calls will return zero length data (indicating end of file) once all previous data has been read.
    + With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the *stream* will fail with *errno* set to EPROTO.
- Any zero length data messages will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop - When the module is popped off the *stream* or the *stream* is closed, the module will take the following action:

- If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

See Also

streamio(M), timod(M), intro(S), getmsg(S), putmsg(S), read(S), write(S), intro(S)

STREAMS Primer
STREAMS Programmer's Guide
Network Programmer's Guide

trchan

translate character sets

Syntax

trchan [-ciko] mapfile

Description

trchan performs mapping as a filter, using the same format of *mapfile* as *mapchan*(M) (described in *mapchan*(F)). This allows a file consisting of one internal character set to be "translated" to another internal character set.

trchan reads standard input, maps it, and writes to standard output. A *mapfile* must be given on the command line. Errors cause *trchan* to stop processing unless **-c** is specified.

The following options can be used with *trchan* :

-c causes errors to be echoed on *stderr*, and processing is continued.

-i specifies that the "input" section of the *mapfile* is used when translating data.

-k specifies that the "dead" and "compose" sections of the *mapfile* are used when translating data.

-o specifies that the "output" section of the *mapfile* is used when translating data.

The **-i**, **-k** and **-o** options can be specified in any combination; if none are specified, *trchan* uses the entire *mapfile*, as if all three were specified together.

Files

/usr/lib/mapchan/*

See Also

ascii(M), mapchan(F), mapchan(M)

Notes

trchan currently ignores the **control** sections of the *mapfile*.

Value Added

trchan is an extension of AT&T System V provided in Altos UNIX System V.

tty

special terminal interface

Description

The file /dev/tty is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired, and when it is tiresome to find out what terminal is currently in use.

The general terminal interface is described in *termio* (M).

Files

/dev/tty
/dev/tty*

See Also

termio(M)

Standards Conformance

tty is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

tz

time zone environment variable

Syntax

/etc/tz

Description

TZ is the shell environment variable for the time zone of the system and is set in the files **/etc/default/login** and **/etc/TIMEZONE** (see *timezone*(F) for a complete description of the syntax for defining TZ).

The shell script **/etc/tz**, generally run during installation, prompts for the correct time zone, prompts for the dates when time is shifted from standard to daylight time and back, and for the number of hours to shift (partial hours in the form of hh:mm:ss are acceptable). and sets TZ correctly in the appropriate files. The following files are examined to see if they read from **/etc/TIMEZONE** to set TZ for their environment:

/etc/cshrc
/etc/profile
/etc/rc2
/.profile

If these files do not read from **/etc/TIMEZONE**, a warning is issued.

Users living in a time zone different than that of the host machine may change *TZ* in their **$HOME/.profile** or **$HOME/.login** files.

To change the time zone for the entire system, run the shell script **/etc/tz** (as root) or use an editor to change the variable *TZ* in the files **/etc/TIMEZONE** and **/etc/default/login**. The *TZ* variable in **/etc/default/login** causes the time zone to be set correctly on logging in and for programs such as *uucico*.

Files

/etc/rc2
/etc/default/login
/etc/tz
$HOME/.profile
$HOME/.login

See Also

environ(M), date(C), timezone(F), ctime(S)

Notes

The *date*(C) automatically switches from Standard Time to Summer Time (Daylight Saving Time). Leap days are properly accounted for.

Changes to *TZ* are immediately effective, (i.e. if a process changes the *TZ* variable, the next call to a *ctime*(S) routine returns a value based on the new value of the variable).

values

machine-dependent values

Syntax

#include <values.h>

Description

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(*type*)	The number of bits in a specified type (e.g., int).
HIBITS	The value of a short integer with only the high-order bit set (in most implementations, 0x8000).
HIBITL	The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT	The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).
MAXLONG	The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).
MAXINT	The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).
MAXFLOAT, LN_MAXFLOAT	The maximum value of a single-precision floating-point number, and its natural logarithm.

MAXDOUBLE, LN_MAXDOUBLE	The maximum value of a double-precision floating-point number, and its natural logarithm.
MINFLOAT, LN_MINFLOAT	The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE, LN_MINDOUBLE	The minimum positive value of a double-precision floating-point number, and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

See Also

intro(S), limits(F), math(M)

Standards Conformance

values is conformant with:

The X/Open Portability Guide II of January 1987.

xtproto

multiplexed channels protocol used by xt(HW) driver

Description

The *xt*(HW) driver contains routines which implement a multiplexed, multi-buffered, full-duplex protocol with guaranteed delivery of ordered data via an 8-bit byte data stream. This protocol is used for communication between multiple UNIX system host processes and an AT&T windowing terminal operating under *layers*(C).

The protocol uses packets with a 2-byte header containing a 3-bit sequence number, 3-bit channel number, control flag, and data size. The data part of a packet may not be larger than 32 bytes. The trailer contains a CRC-16 code in 2 bytes. Each channel is double-buffered.

Correctly received packets in sequence are acknowledged with a control packet containing an ACK; however, out of sequence packets generate a control packet containing a NAK, which will cause the retransmission in sequence of all unacknowledged packets.

Unacknowledged packets are retransmitted after a timeout interval which is dependent on baud rate. Another timeout parameter specifies the interval after which incomplete receive packets are discarded.

Files

/usr/include/sys/xtproto.h channel multiplexing protocol definitions

See Also

layers(M), layers(C), xt(HW)

Altos UNIX® System V/386 Release 3.2

(F) File Formats

Contents

File Formats (F)

maxuuxqts	UUCP uuxqt(C) limit file
mdevice	device driver module description file
meisa	master system kernel configuration file
mem, kmem	memory image file
memtune	table of tunable parameters to be adjusted when adding more memory
mfsys	configuration file for filesystem types
micnet	the Micnet default commands file
mmdftailor	provides run-time tailoring for the MMDF mail router
mnttab	format of mounted filesystem table
mtune	tunable parameter file
mvdevice	video driver backend configuration file
nl_types	data types for native language support
null	the null file
passwd	the password file
permissions	format of UUCP Permissions file
plot	graphics interface
pnch	file format for card images
poll	format of UUCP Poll file
purge	the policy file of the sanitization utility purge(C)
queue	MMDF queue files for storing mail in transit
queuedefs	scheduling information for cron queues
reloc	relocation information for a common object file
sccsfile	format of an SCCS file
scnhdr	section header for a common object file
scr_dump	format of curses screen image file
sdevice	local device configuration file
sfsys	local filesystem type file
stat	data returned by stat system call
stune	local tunable parameter file
syms	common object file symbol table format
sysfiles	format of UUCP Sysfiles file
systemid	the Micnet system identification file
systems	format of UUCP Systems file
tables	MMDF name tables for aliases, domains, and hosts
tar	archive format
term	terminal driving tables for nroff
termcap	terminal capability data base
terminfo	format of compiled terminfo file
timezone	set default system time zone
top, top.next	the Micnet topology files
types	primitive system data types
unistd	file header for symbolic constants
utmp, wtmp	formats of utmp and wtmp entries

Intro

Introduction to file formats

Description

This section outlines the formats of various files. The C **struct** declarations for the file formats are given where applicable. Usually, these structures can be found in the directories **/usr/include** or **/usr/include/sys**. Note that include files are part of the Development System.

86rel

Intel 8086 Relocatable Format for Object Modules

Syntax

#include <sys/relsym86.h>

Description

Intel 8086 Relocatable Format, or *86rel*, is the object module format generated by *masm*(CP), and the input format for the linker *ld*(CP). The include file **relsym86.h** specifies appropriate definitions to access *86rel* format files from C. For the technical details of the *86rel* format, see *Intel 8086 Object Module Format External Product Specification*.

An *86rel* consists of one or more variable length records. Each record has at least three fields: the record type, length, and checksum. The first byte always denotes the record type. There are thirty-one different record types. Only eleven are used by *ld*(CP) and *masm*(CP). The word after the first byte is the length of the record in bytes, exclusive of the first three bytes. Following the length word are typically one or more fields. Each record type has a specific sequence of fields, some of which may be optional or of varying length. The very last byte in each record is a checksum. The checksum byte contains the sum modulo 256 of all other bytes in the record. The sum modulo 256 of all bytes in a record, including the checksum byte, should equal zero.

With few exceptions, *86rel* strings are length prefixed and have no trailing null. The first byte contains a number between 0 and 40, which is the remaining length of the string in bytes. Although the Intel specification limits the character set to upper case letters, digits, and the characters "?", "@", ":", ".", and "_", *masm*(CP) uses the complete ASCII character set.

The Intel Object Module Format (OMF) specification uses the term "index" to mean a positive integer either in the range 0 to 127, or 128 to 32,768. This terminology is retained in this document and elsewhere in the *86rel* literature. An index has one or two bytes. If the first byte has a leading 0 bit, the index is assumed to have only one byte, and the remainder of the byte represents a positive integer between 0 and 127. If the second byte has a leading 1 bit, the index is assumed to take up two bytes, and the remainder of the word represents a positive integer between 128 and 32,768.

Following is a list of record types and the hexadecimal value of their first byte, as defined in **relsym86.h.**

```
#define MRHEADR    0x6e /*rel module header/*
#define MREGINT       0x70 /*register initialization*/
#define MREDATA    0x72 /*explicit (enumerated) data image*/
#define MRIDATA       0x74 /*repeated (iterated) data image*/
#define MOVLDEF       0x76 /*overlay definition*/
#define MENDREC       0x78 /*block or overlay end record*/
#define MBLKDEF       0x7a /*block definition*/
#define MBLKEND       0x7c /*block end*/
#define MDEBSYM       0x7e /*debug symbols*/
#define MTHEADR    0x80 /*module header,
                            *usually first in a rel file*/
#define MLHEADR    0x82 /*link module header*/
#define MPEDATA    0x84 /*absolute data image*/
#define MPIDATA       0x86 /*absolute repeated (iterated)
                            *data image*/
#define MCOMENT    0x88 /*comment record*/
#define MMODEND    0x8a /*module end record*/
#define MEXTDEF       0x8c /*external definition*/
#define MTYPDEF       0x8e /*type definition*/
#define MPUBDEF       0x90 /*public definition*/
#define MLOCSYM       0x92 /*local symbols*/
#define MLINNUM       0x94 /*source line number*/
#define MLNAMES    0x96 /*name list record*/
#define MSEGDEF       0x98 /*segment definition*/
#define MGRPDEF       0x9a /*group definition*/
#define MFIXUPP       0x9c /*fix up previous data image*/
#define MNONE1     0x9e /*none*/
#define MLEDATA    0xa0 /*logical data image*/
#define MLIDATA       0xa2 /*logical repeated (iterated)
                            *data image*/
#define MLIBHED       0xa4 /*library header*/
#define MLIBNAM       0xa6 /*library names record*/
#define MLIBLOC       0xa8 /*library module locations*/
#define MLIBDIC       0xaa /*library dictionary*/
#define M386END       0x86 /*32 bit module end record*/
#define MPUB386       0x91 /*32 bit public definition*/
#define MLOC386       0x93 /*32 bit logical symbols*/
#define MLIN386       0x95 /*32 bit source line number*/
#define MSEG386       0x99 /*32 bit segment definition*/
#define MFIX386       0x9d /*fix up previous 32 bit data image*/
#define MLED386       0xa1 /*32 bit logical data image*/
#define MLID386       0xa3 /*32 bit logical repeated (iterated) data image*/
```

In the following discussion, the salient features of each record type are given. If the record is not used by either *masm*(CP) or *ld*(CP), it is not listed.

THEADR — The record type byte is 0x80. The THEADR record specifies the name of the source module at assembly-time (see Notes). The sole field is the T-MODULE NAME , which contains a length-prefixed string derived from the base name of the source module.

COMENT
: The record type byte is 0x88. The COMENT record may contain a remark generated by the compiler system. *masm*(CP) inserts the string "UNIX 8086 ASSEMBLER."

MODEND
: The record type byte is 0x8a. The MODEND record terminates a module. It can specify whether the current module is to be used as the entry point to the linked executable. If the module is an entry point, the MODEND record can then specify the address of the entry point within the executable.

EXTDEF
: The record type byte is 0x8c. The EXTDEF record contains the names and types of symbols defined in other modules by a PUBDEF record (see below). This corresponds to the C storage class "extern." The fields consist of one or more length-prefixed strings, each with a following type index. The indices reference a TYPDEF record seen earlier in the module. *masm*(CP) generates only one EXTDEF per exterior symbol.

TYPDEF
: The record type byte is 0x8e. The TYPDEF record gives a description of the type (size and storage attributes) of an object or objects. This description can then be referenced by EXTDEF , PUBDEF , and other records.

PUBDEF
: The record type byte is 0x90. The PUBDEF record gives a list of one or more names that may be referenced by other modules at link-time ("publics"). The list of names is preceded by a group and segment index, which reference the location of the start of the list of publics within the current segment and group. If the segment and group indices are zero, a frame number is given to provide an absolute address in the module. The list consists of one or more of length-prefixed strings, each associated with a 16-bit offset within the current segment and a type index referring to a TYPDEF .

LNAMES
: The record type byte is 0x96. The LNAMES record gives a series of length-prefixed strings which are associated with name indices within the current module. Each name is indexed in sequence given starting with 1. The names may then be referenced within the current module by successive SEGDEF and GRPDEF records to provide strings for segments, classes, overlays or groups.

SEGDEF
: The record type byte is 0x98. The SEGDEF record provides an index to reference a segment, and information concerning segment addressing and attributes. This index may be used by other records to refer to the segment. The first byte in the record after the length field gives information about the alignment, and about combination attributes of the segment. The next word is the segment length in bytes. Note that this restrains segments to a maximum 65,536 bytes in length. Following this word is an index (see above) for the segment. Lastly, the SEGDEF may optionally contain class and/or overlay index fields.

GRPDEF
: The record type is 0x9a. The GRPDEF record provides a name to reference several segments. The group name is implemented as an index (see above).

FIXUPP
: The record byte is 0x9c. The FIXUPP record specifies one or more load-time address modifications ("fixups"). Each fixup refers to a location in a preceding LEDATA (see below) record. The fixup is specified by four data; a location, a mode, a target and a frame. The frame and target may be specified explicitly or by reference to an already defined fixup.

LEDATA
: The record type byte is 0xa0. This record provides a contiguous text or data image which the loader *ld*(CP) uses to construct a portion of an 8086 run-time executable. The image might require additional processing (see FIXUPP) before being loaded into the executable. The image is preceded by two fields, a segment index and an enumerated data offset. The segment index (see INDEX) specifies a segment given by a previously seen SEGDEF . The enumerated data offset (a word) specifies the offset from the start of this segment.

See Also

as(CP), ld(CP)

Notes

If you attempt to load a number of modules assembled under the same basename, the loader will try to put them all in one big segment. In 286 programs, segment size is limited to 64K. In a large program the resulting segment size can easily exceed 64K. A large model code executable results from the link of one or more modules, composed of segments that aggregate into greater than 64K of text.

Hence, be sure that the assembly-time name of the module has the same basename as the source. This can occur if the source module is preprocessed not by *cc*(CP), but, for example, by hand or shell script, prior to assembly. The following example is incorrect:

```
#incorrect
cc -E module1.c | filter > x.c
cc x.c
mv x.o module1.o
cc -E module2.c | filter > x.c
cc x.c
mv x.o module2.o
cc -E module3.c | filter > x.c
cc x.c
mv x.o module3.o
ld module1.o module2.o module3.o
```

To avoid this, each of the modules should have a unique name when assembled, as follows:

```
#correct
cc -E module1.c | filter > x.c
cc -S x.c
mv x.s module1.s
as module1.s
.
.
.
ld module1.o module2.o module3.o
```

Value Added

86rel is an extension of AT&T System V provided in Altos UNIX System V.

a.out

UNIX common assembler and link editor output

Syntax

#include <a.out.h>

Description

The file name **a.out** is the default output file name from the link editor *ld*(CP). The link editor will make *a.out* executable if there were no errors in linking. The output file of the UNIX assembler *as (CP)* also follows the common object file format of the *a.out* file although the default file name is different.

A common object file consists of a file header, a UNIX system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
UNIX system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **-s** option of *ld*(CP) or if they were removed by *strip*(CP). Also note that the relocation information will be absent after linking unless the **-r** option of *ld*(CP) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. On your computer, the text segment starts at location virtual address 0.

The a.out file produced by *ld*(CP) may have one of two magic numbers in the first field of the UNIX system header. A magic number of 0410 indicates that the executable must be swapped through the private swapping store of the UNIX system, while the magic number 0413 causes the system to attempt to page the text directly from the a.out file.

In a 0410 executable, the text section is loaded at virtual location 0x00000000. The data section is loaded immediately following the end of the text section.

For a 0413 executable, the headers (file header, UNIX system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address will equal the sum of the sizes of the headers, and will vary depending on the number of sections in the a.out file. In an a.out file with 3 sections (.text, .data, and .bss) the first text address is at 0x000000D0. The data section starts in the next page table directory after the last one used by the text section, in the first page of that directory, with an offset into that page equal to the 1st unused memory offset in the last page of text. That is to say, given that *etext* is the address of the last byte of the text section, the 1st byte of the data section will be at 0x00400000 + (*etext* & 0xFFC00000) + ((*etext*+1) & 0xFFC00FFF).

On the 80386 computer the stack begins at location 7FFFFFFC and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the *brk*(S) system call.

For relocatable files the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an "external symbol", and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

File Header

The format of the **filehdr** header is:

```
struct filehdr
{
        unsigned short  f_magic;    /* magic number */
        unsigned short  f_nscns;    /* number of sections */
        long            f_timdat;   /* time and date stamp */
        long            f_symptr;   /* file ptr to symtab */
        long            f_nsyms;    /* # symtab entries */
        unsigned short  f_opthdr;   /* sizeof(opt hdr) */
        unsigned short  f_flags;    /* flags */
};
```

UNIX System Header

The format of the UNIX system header is:

```
typedef struct aouthdr
{
        short   magic;       /* magic number */
        short   vstamp;      /* version stamp */
        long    tsize;       /* text size in bytes, padded */
        long    dsize;       /* initialized data (.data) */
        long    bsize;       /* uninitialized data (.bss) */
        long    entry;       /* entry point */
        long    text_start;  /* base of text used for this file */
        long    data_start;  /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is:

```
struct scnhdr
{
        char            s_name[SYMNMLEN];/* section name */
        long            s_paddr;    /* physical address */
        long            s_vaddr;    /* virtual address */
        long            s_size;     /* section size */
        long            s_scnptr;   /* file ptr to raw data */
        long            s_relptr;   /* file ptr to relocation */
        long            s_lnnoptr;  /* file ptr to line numbers */
        unsigned short  s_nreloc;   /* # reloc entries */
        unsigned short  s_nlnno;    /* # line number entries */
        long            s_flags;    /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```
struct reloc
{
        long       r_vaddr;    /* (virtual) address of reference */
        long       r_symndx;   /* index into symbol table */
        ushort     r_type;     /* relocation type */
};
```

The start of the relocation information is *s_relptr* from the section header. If there is no relocation information, *s_relptr* is 0.

Symbol Table

The format of each symbol in the symbol table is

```
#define  SYMNMLEN 8
#define  FILNMLEN 14
#define  DIMNUM   4

struct syment
{
   union                            /* all ways to get a symbol name */
   {
      char         _n_name[SYMNMLEN]; /* name of symbol */
      struct
      {
         long      _n_zeroes;       /* == 0L if in string table */
         long      _n_offset;       /* location in string table */
      } _n_n;
      char         *_n_nptr[2];     /* allows overlaying */
   } _n;
   long            n_value;         /* value of symbol */
   short           n_scnum;         /* section number */
   unsigned short  n_type;          /* type and derived type */
   char            n_sclass;        /* storage class */
   char            n_numaux;        /* number of aux entries */
};

#define  n_name    _n._n_name
#define  n_zeroes  _n._n_n._n_zeroes
#define  n_offset  _n._n_n._n_offset
#define  n_nptr    _n._n_nptr[1]
```

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent {
        struct {
                long    x_tagndx;
                union {
                        struct {
                                unsigned short  x_lnno;
                                unsigned short  x_size;
                        } x_lnsz;
                        long      x_fsize;
                } x_misc;
                union {
                        struct {
                                long   x_lnnoptr;
                                long   x_endndx;
                        }  x_fcn;
                        struct {
                                unsigned short  x_dimen[DIMNUM];
                        } x_ary;
                } x_fcnary;
                unsigned short  x_tvndx;
        } x_sym;

        struct {
                char    x_fname[FILNMLEN];
        } x_file;

        struct {
                long        x_scnlen;
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        } x_scn;

        struct {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        } x_tv;
};
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is *f_symptr* (from the file header) bytes from the beginning of the file. If the symbol table is stripped, *f_symptr* is 0. The string table (if one exists) begins at *f_symptr* + (*f_nsyms* * SYMESZ) bytes from the beginning of the file.

See Also

as(CP), cc(CP), ld(CP), brk(S). filehdr(F), ldfcn(F), linenum(F), reloc(F), scnhdr(F), syms(F).

acct

format of per-process accounting file

Description

Files produced as a result of calling *acct*(S) have records in the form defined by **<sys/acct.h>.**

In *ac_flag*, the AFORK flag is turned on by each *fork*(S) and turned off by an *exec*(S). The *ac_comm* field is inherited from the parent process and is reset by any *exec*. Each time the system charges the process with a clock tick, it also adds the current process size to *ac_mem* computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of *ac_mem*/*ac_stime* can be viewed as an approximation to the mean process size, as modified by text-sharing.

See Also

acctcom(ADM), acct(S)

Notes

The *ac_mem* value for a short-lived command gives little information about the actual size of the command, because *ac_mem* may be incremented while a different command (e.g., the shell) is being executed by the process.

Standards Conformance

acct is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ar

archive file format

Description

The archive command *ar* is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor *ld*(C).

A file produced by *ar* has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number is 0177545 octal (or 0xff65 hexadecimal). The header of each file is declared in **/usr/include/ar.h**.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

See Also

ar(CP), ld(CP)

archive

default backup device information

Description

The **/etc/default/archive** file contains information on system default backup devices for use by *sysadmsh*(ADM). The device entries are in the following format:

name=value [name=value] ...

value may contain white spaces if quoted, and newlines may be escaped with a backslash.

The following names are defined for **/etc/default/archive**:

bdev	Name of the block interface device.
cdev	Name of the character interface device.
size	Size of the volume in either blocks or feet.
density	Volume density, such as 1600. If this value is missing or null, then *size* is in blocks; otherwise the *size* is in feet.
format	Command used to format the archive device.
blocking	Blocking factor.
desc	A description of the device, such as ''Cartridge Tape.''

See Also

sysadmsh(ADM)

authcap

authentication database

Syntax

/etc/auth/*
/tcb/files/auth/*

Description

The database contains authentication and identity information for users, kernels, and TCB files as well as system-wide parameters. It is intended to be used by programs to interrogate user and system values, as well as by authentication programs to update that information.

Structure of the Hierarchies

The complete database resides in two hierarchies: **/tcb/files/auth** and **/etc/auth**. The first hierarchy deals with user-specific files, and has subdirectories of one letter each of which is the starting letter for user name. Within each of these directories are regular files, each containing an *authcap*(F) format file for a particular user. Thus, all user names beginning with **x** have their respective authentication and identity information in a file in directory **/tcb/files/auth/x**.

The directories within **/etc/auth** contain system-wide information. The global system settings reside in the **/etc/auth/system** directory. The subsystem authorizations associated with each protected subsystem (a protected subsystem is privileged but does not require global authority to perform actions) are located in the **/etc/auth/subsystems** directory.

The following database files are contained in the **system** directory:

default	Default Control
files	File Control
ttys	Terminal Control
authorize	Primary and Secondary Authorization Control File
devassign	Device Assignment

A subsystem file name is the group name associated with the protected subsystem. The owner of all files is **auth** and the group is the group of the subsystem. Only the owner and group of this file may view the contents. The file **dflt_users** lists the users granted default subsystem authorizations.

Format of a File

Each data file in the hierarchy, whether system-wide or user-specific, has the same format. Each user file consists of one virtual line, optionally split into multiple physical lines with the '/' character present at the very end of all lines but the last. For instance, the line

```
blf:u_name=blf:u_id#16:u_encrypt=a78/a1.eitfn6:u_type=sso:chkent:
```

may be split into:

```
blf:u_name=blf:u_id#16:\
        :u_encrypt=a78/a1.eitfn6:\
        :u_type=sso:chkent:
```

Note that all capabilities must be immediately preceded and followed with the ':' separator; multiple line entries require additional ones - one more per line. Multiple entries are separated by a newline:

```
drb:u_name=drb:u_id#75:u_maxtries#9:u_type=general:chkent:
blf:u_name=blf:u_id#76:u_maxtries#5:u_type=general:chkent:
```

For subsystem files, the file is a set of lines, each containing a user name terminated by a colon, followed by a comma-separated list of primary and secondary authorizations defined for that subsystem.

Format of a Line

The format of a line (except for subsystem files) is briefly as follows:

name|alt name(s)|description:cap1:cap2:cap3:...:capn:chkent:

The entry can be referenced by the name or any of the alternate names. A description field may document the entry. The entry name(s) and description are separated by the '|' character. The end of the name/description part of the entry is terminated by the ':' character. Alternate names and the description fields are optional.

At the end of each entry is the *chkent* field. This is used as an integrity check on each entry. The *authcap*(S) routines will reject all entries that do not have *chkent* at the very end.

Each entry has 0 or more capabilities, each terminated with the ':' character. Each capability has a unique name. Numeric capabilities have the format:

id#*num*

where num is a decimal or (0 preceded) octal number. Boolean capabilities have the format:

id

or

id@

where the first form signals the presence of the capability and the second form signals the absence of the capability. String capabilities have the format:

id=*string*

where string is 0 or more characters. The '\' and ':' characters are escaped as '\\' and '\:' respectively. Although it is not recommended, the same id may be used for different numeric, boolean, and string capabilities.

See Also

getprpwent(S), getdvagent(S), getprtcent(S), getprfient(S)

Value Added

authcap is an extension of AT&T System V provided in Altos UNIX System V.

card_info

system tty controller card information file

Description

The **card_info** file is used by *pcu*(ADM) to map tty *inittab*(F) entries to installed tty controller cards. Boards that control both serial and parallel ports and that use different drivers to support these ports will have two entries in the **card_info** file.

Each line must contain the following seven fields, each separated by a colon:

ID:*port_type*:*board_name*:*regexp*:*format*:*range*:*package_id*

The first five fields are mandatory. The sixth and seventh fields are only used by ISA tty controller card entries and should be blank for EISA tty controllers. Each field is described below:

ID This field contains the uncompressed EISA identifier for the board. A full seven-character ID (for example, ACS0301) can be used, but the minor revision number may be omitted (for example, ACS030). For ISA tty controller boards field one contains the word **ISA**.

port_type
This field contains the type of the ports on the board or the EISA subfunction type string. This string can be a simple EISA type (e.g. COM,MDC), or may include subtype information (e.g., COM,ASY;COM1). In addition, an abbreviated type string can be used. For example, COM,ASY will match both COM,ASY;COM1 and COM,ASY;COM2 on the specified board. For ISA tty controller boards this field should contain one of the following types:

COM - Indicates that this is a normal serial communications port.

PAR - Indicates that this is a parallel printer port.

board_name
This field contains the full name of the board which is displayed for the user in the title bar of the ports window in **pcu**.

regexp
This field contains a *regex*(S) pattern to identify the ports that belong to this board or subfunction. This *regex* pattern returns a value (usually numeric) that distinguishes one port on the board

from the next. For example, if the pattern **tty([0-9]{2,4})$0** was given the port name **tty03**, the **03** would be returned and *regex* would succeed.

format

This field contains a *printf*(S) format string for printing a port's ttyname and takes as its single argument the value returned by the *regexp* pattern from field four. For example, the format string **tty%02d** would be used with the *regexp* pattern example given above. Also note that the format string should be for the non-modem tty port name. For example, for com1/com2 ports use **tty%da** not **tty%dA**.

range

This field contains a range of numbers of the form *n1-n2* used as arguments to the format string in field five to generate the ttynames for the non-modem ports attached to the board. This field is for ISA entries only. For EISA entries *pcu* gets the port range from the **slot_info** file, which is created by *uconfig* and its execution of **/etc/conf/pack.d/*/*node** scripts.

package_id

This field contains the device driver device name for the board as found in the first field of the device driver's *mdevice*(F) file entry. This field is for ISA entries only. For EISA entries *pcu* gets the package_id from **slot_info**, which is created by *uconfig*.

Files

/etc/card_info
/etc/slot_info

See Also

pcu(ADM), uconfig(ADM)

Value Added

card_info is an extension to AT&T UNIX System V provided in Altos UNIX System V.

checklist

list of file systems processed by fsck(ADM)

Description

The **/etc/checklist** file contains a list of the file systems to be checked when *fsck*(ADM) is invoked without arguments. The list contains at most 15 **special file** names. Each **special file** name must be on a separate line and must correspond to a file system.

See Also

fsck(ADM)

clock

the system real-time (time of day) clock

Description

The **clock** file provides access to the battery-powered, real-time time of day clock. Reading this file returns the current time; writing to the file sets the current time. The time, 10 bytes long, has the following form:

MMddhhmmyy

where *MM* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *yy* is the last two digits of the year. For example, the time:

0826150389

is 15:03 (3:03 PM) on August 26, 1989.

Files

/dev/clock

See Also

setclock(ADM)

core

format of core image file

Description

The Operating System writes out a core image of a terminated process when any of various errors occur. See *signal*(S) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called *core* and is written in the process' working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The first section of the core image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter *usize*, which is defined in **/usr/include/sys/param.h**. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in **/usr/include/sys/user.h**. The locations of registers, are outlined in **/usr/include/sys/reg.h**.

See Also

adb(CP), setuid(S), signal(S)

cpio

format of cpio archive

Description

The *header* structure, when the **-c** option of *cpio*(C) is not used, is:

```
struct {
            short     h_magic,
                      h_dev;
            ushort    h_ino,
                      h_mode,
                      h_uid,
                      h_gid;
            short     h_nlink,
                      h_rdev,
                      h_mtime[2],
                      h_namesize,
                      h_filesize[2];
            char      h_name[h_namesize rounded to word];
} Hdr;
```

When the **-c** option is used, the *header* information is described by:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
        &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
        &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
        &Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The contents of each file are recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in *stat*(S). The length of the null-terminated path name *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with *h_filesize* equal to zero.

See Also

cpio(C), find(C), stat(S)

Standards Conformance

cpio is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

default

default program information directory

Description

The files in the directory **/etc/default** contain the default information used by system commands such as **xbackup**(ADM) and **remote**(C). Default information is any information required by the command that is not explicitly given when the command is invoked.

The directory may contain zero or more files. Each file corresponds to one or more commands. A command searches a file whenever it has been invoked without sufficient information. Each file contains zero or more entries which define the default information. Each entry has the form:

keyword

or

keyword=value

where *keyword* identifies the type of information available and *value* defines its value. Both *keyword* and *value* must consist of letters, digits, and punctuation. The exact spelling of a *keyword* and the appropriate *values* depend on the command and are described with the individual commands.

Any line in a file beginning with a number sign (#) is considered a comment and is ignored.

Files

/etc/default/*

See Also

archive(F), xbackup(ADM), boot(HW), cron(C), dos(C), dumpdir(C), filesys(F), login(M), lpr(C), mapchan(M), mapchan(F), micnet (F), authsh(ADM) remote(C), xrestore(ADM), su(C), tar(C)

Note

Not all commands use **/etc/default** files. Please refer to the manual page for a specific command to determine if **/etc/default** files are used, and what information is specified.

Value Added

default is an extension of AT&T System V provided by the Santa Cruz Operation.

devices

format of UUCP devices file

Description

The **Devices** file (**/usr/lib/uucp/Devices**) contains information for all the devices that can be used to establish a link to a remote computer. These devices include automatic call units, direct links, and network connections. This file works closely with the **Dialers**, **Systems**, and **Dialcodes** files.

Each entry in the **Devices** file has the following format:

type ttyline dialerline speed dialer-token

where:

type can contain one of two keywords (**direct** or **ACU**), the name of a Local Area Network switch, or a system name.

ttyline contains the device name of the line (port) associated with the *Devices* entry. For example, if the Automatic Dial Modem for a particular entry is attached to the **/dev/tty11** line, the name entered in this field is **tty11**.

dialerline is useful only for 801 type dialers, which do not contain a modem and must use an additional line. If you do not have an 801 dialer, enter a hyphen (-) as a placeholder.

speed is the speed or speed range of the device. It may contain an indicator for distinguishing different dialer classes.

dialer-token contains pairs of dialers and tokens. Each represents a dialer and an argument to be passed to it. The *dialer* portion can be the name of an automatic dial modem, or it may be a **direct** for a direct link device.

For best results, dialer programs are preferred over **Dialers** entries. The following entry is an example of an entry using a dialer binary:

```
ACU  ttynn  -  300-2400  /usr/lib/uucp/dialHA24
```

Note all lines must have at least 5 fields. Use "-" for unused fields. Types that appear in the 5th field must be either built-in functions (801, Sytek, TCP, Unetserver, DK) or standard functions whose name appears in the first field in the **Dialers** file.

Two escape characters can be used in this file:

\D　which means don't translate the phone /token

\T　translate the phone /token using the Dialcodes file

Both refer to the phone number field in the **Systems** file (field 5). \D should always be used with entries in the **Dialers** file, since the **Dialers** file can contain a T to expand the number if necessary. \T should only be used with built-in functions that require expansion.

Note that if a phone number is expected and a \D or \T is not present a \T is used for a built-in, and \D is used for an entry referencing the **Dialers** file.

Examples

The following are examples of common **Devices** files.

Standard modem line

```
ACU tty00 - 1200 801
ACU tty00 - 1200 penril
or
ACU tty00 - 1200 penril \D
```

A direct line

This example will allow **cu -ltty00** to work. This entry could also be used for certain modems in manual mode.

```
Direct tty00 - 4800 direct
```

A ventel modem on a develcon switch

"vent" is the token given to the develcon to reach the ventel modem.

```
ACU tty00 - 1200 develcon vent ventel
ACU tty00 - 1200 develcon vent ventel \D
```

To reach a system on the local develcon switch

```
Develcon tty00 - Any develcon \D
```

A direct connection to a system

```
systemx tty00 - Any direct
```

STREAMS Network Examples

A STREAMS network that conforms to the AT&T Transport Interface with a direct connection to login service (i.e., without explicitly using the Network Listener Service dial script):

```
networkx,eg devicex - - TLIS \D
```

The **Systems** file entry looks like:

```
systemx Any networkx - addressx in:--in: nuucp word: nuucp
```

You must replace *systemx*, *networkx*, *addressx*, and *devicex* with system name, network name, network address and network device, respectively. For example, entries for machine ''sffo'' on a STARLAN NETWORK might look like:

```
sffoo Any STARLAN - sffoo in:--in: nuucp word: nuucp
```

and:

```
STARLAN,eg starlan - - TLIS \D
```

To use a STREAMS network that conforms to the AT&T Transport Interface and that uses the Network Listener Service dial script to negotiate for a server:

```
networkx,eg devicex - - TLIS \D nls
```

To use a non-STREAMS network that conforms to the AT&T Transport Interface and that uses the Network Listener Service dial script to negotiate for a server:

```
networkx,eg devicex - - TLI \D nls
```

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C), dialers(F)

Notes

Blank lines and lines that begin with a <space>, <tab>, or are ignored. protocols can be specified as a comma-subfield of the device type either in the **Devices** file (where device type is field 1) or in the **Systems** file (where it is field 3).

dialcodes

format of UUCP Dialcode abbreviations file

Description

The **Dialcodes** file (**/usr/lib/uucp/Dialcodes**) contains the Dialcode abbreviations that can be used in the *Phone* field of the **Systems** file. This feature allows you to create a standard **Systems** file for distribution among several sites that have different phone systems and area codes.

If two remote sites in a network need to link with the same sites, but have different internal phone systems each site can share the same **Systems** file, but have different entries in a **Dialcodes** file. Each entry has the following format:

abb dial-seq

where:

abb is the abbreviation used in the **Systems** file *phone* field

dial-seq is the dial sequence that is passed to the dialer when that particular **Systems** file entry is accessed.

The following entry would be set up to work with a *phone* field in the *Systems* file such as *jt7867* :

```
jt 9=847-
```

When the entry containing *jt7867* is encountered, the following sequence is sent to the dialer if the token in the dialer-token-pair is \T :

```
9=847-7867
```

The phone number is made up of an optional alphabetic abbreviation and a numeric part. If an abbreviation is used, it must be one that is listed in the *Dialcodes* file.

```
NY 9=1212555
```

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C), systems(F)

dialers

format of UUCP Dialers file

Description

The **Dialers** file (**/usr/lib/uucp/Dialers**) specifies the initial conversation that must take place on a line before it can be made available for transferring data. This conversation is usually a sequence of ASCII strings that is transmitted and expected, and it is often used to dial a phone number using an ASCII dialer (such as the Automatic Dial Modem).

A modem that is used for dialing in and out may require a second **Dialers** entry. This is to reinitialize the line to dial-in after it has been used for dial-out. The name of the dial-in version of a dialer must begin with an ampersand. For example, the **Dialers** file contains a **hayes2400** and a **&hayes2400** entry.

The fifth field in a **Devices** file entry is an index into the **Dialers** file or a special dialer type. Here an attempt is made to match the fifth field in the **Devices** file with the first field of each **Dialers** file entry. In addition, each odd numbered **Devices** field starting with the seventh position is used as an index into the **Dialers** file. If the match succeeds, the **Dialers** entry is interpreted to perform the dialer negotiations. Each entry in the **Dialers** file has the following format:

dialer substitutions expect-send ...

The *dialer* field matches the fifth and additional odd numbered fields in the **Devices** file. The *substitutions* field is a translate string: the first of each pair of characters is mapped to the second character in the pair. This is usually used to translate = and - into whatever the dialer requires for ``wait for dialtone'' and ``pause.''

The remaining *expect-send* fields are character strings. Below are some character strings distributed with the UUCP package in the **Dialers** file.

Dialers file entries

```
penril     =W-P "" \d > s\p9\c )-W\p\r\ds\p9\c-) y\c : \E\TP > 9\c OK
ventel     =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
hayes      =,-, "" \dAT\r\c OK\r \EATDT\T\r\c CONNECT
rixon      =&-% "" \d\r\r\c $ s9\c )-W\r\ds9\c-) s\c : \T\r\c $ 9\c LINE
vadiac     =K-K "" \005\p *-\005\p-*\005\p-* D\p BER? \E\T\e \r\c LINE
develcon   "" "" \pr\ps\c est:\007 \E\D\e \007
micom      ""    "" \s\c NAME? \D\r\c GO
direct
att2212c   =+-,    "" \r\c :--: ato12=y,T\T\r\c red
att4000    =,-,    "" \033\r\r\c DEM: \033s0401\c \006 \033s0901\c \
               \006 \033s1001\c \006 \033s1102\c \006 \033dT\T\r\c \006
att2224    =+-,    "" \r\c :--: T\T\r\c red
nls           ""      "" NLPS:000:001:1\N\c
```

The meaning of some of the escape characters (those beginning with "\") used in the **Dialers** file are listed below:

\p	pause (approximately ¼ to ½ second)
\d	delay (approximately 2 seconds)
\D	phone number or token without **Dialcodes** translation
\T	phone number or token with **Dialcodes** translation
\K	insert a BREAK
\E	enable echo checking (for slow devices)
\e	disable echo checking
\r	carriage return
\c	no new-line or carriage return
\n	send new-line
\nnn	send octal number.

Additional escape characters that may be used are listed in the section discussing the **Systems** file.

The penril entry in the **Dialers** file is executed as follows. First, the phone number argument is translated, replacing any = with a **W** (wait for dialtone) and replacing any - with a **P** (pause). The handshake given by the remainder of the line works as follows:

""	Wait for nothing.
\d	Delay for 2 seconds.
>	Wait for a >.

`s\p9\c`	Send an **s**, pause for ½ second, send a **9**, send no terminating new-line
`)-W\p\r\ds\p9\c-)`	Wait for a **)**. If it is not received, process the string between the **-** characters as follows. Send a **W**, pause, send a carriage-return, delay, send an **s**, pause, send a **9**, without a new-line, and then wait for the **)**.
`y\c`	Send a **y**.
`:`	Wait for a **:**.
`\E\TP`	Enable echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.) Then, send the phone number. The **\T** means take the phone number passed as an argument and apply the **Dialcodes** translation and the modem function translation specified by field 2 of this entry. Then send a **P**.
`>`	Wait for a **>**.
`9\c`	Send a **9** without a new-line.
`OK`	Waiting for the string **OK**.

See Also

dial(ADM), uucico(ADM), uucp(C), uux(C), uuxqt(C), devices(F)

Notes

Dialer binaries (located in **/usr/lib/uucp**) are preferred over **Dialers** entries. Binaries are more reliable. Refer to the *dial* man page for more information on creating your own dialer binaries.

dir

format of a directory

Syntax

#include <sys/dir.h>

Description

A directory behaves exactly like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry (see *filesystem*(F)). The structure of a directory is given in the include file **/usr/include/sys/dir.h.**

By convention, the first two entries in each directory are''dot'' (.) and ''dotdot'' (..). The first is an entry for the directory itself. The second is for the parent directory. The meaning of dotdot is modified for the root directory of the master file system; there is no parent, so dotdot has the same meaning as dot.

See Also

filesystem(F)

dirent

filesystem-independent directory entry

Syntax

#include <sys/types.h>
#include <sys/dirent.h>

Description

Different file system types may have different directory entries. The *dirent* structure defines a file-system-independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the *getdents* (S) system call.

The *dirent* structure is defined below.

```
struct  dirent {
                        long                d_ino;
                        off_t               d_off;
                        unsigned short      d_reclen;
                        char                d_name[1];
            };
```

The *d_ino* is a number which is unique for each file in the file system. The field *d_off* is the offset of that directory entry in the actual file system directory. The field *d_name* is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most MAXNAMLEN characters. This results in file system independent directory entries being variable length entities. The value of *d_reclen* is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that it will always result in the next entry being on a long boundary.

Files

/usr/include/sys/dirent.h

See Also

getdents(S)

Standards Conformance

dirent is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

filehdr

file header for common object files

Syntax

#include <filehdr.h>

Description

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct  filehdr
{
        unsigned short  f_magic ;    /* magic number */
        unsigned short  f_nscns ;    /* number of sections */
        long            f_timdat ;   /* time & date stamp */
        long            f_symptr ;   /* file ptr to symtab */
        long            f_nsyms ;    /* # symtab entries */
        unsigned short  f_opthdr ;   /* sizeof(opt hdr) */
        unsigned short  f_flags ;    /* flags */
} ;
```

F_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek*(S) to position an I/O stream to the symbol table. The UNIX system optional header is 28-bytes. The valid magic numbers are given below:

```
#define I286SMAGIC 0512 /* 80286 computers—small model
                           programs */
#define I286LMAGIC 0522 /* 80286 computers—large model
                           programs */
#define I386MAGIC  0514 /* 80386 computers */
#define FBOMAGIC   0560 /* 3B2 and 3B15 computers */
#define N3BMAGIC   0550 /* 3B20 computer */
#define NTVMAGIC   0551 /* 3B20 computer */

#define VAXWRMAGIC 0570 /* VAX writable text segments */
#define VAXROMAGIC 0575 /* VAX read only sharable
                           text segments */
```

The value in *f_timdat* is obtained from the *time*(S) system call. Flag bits currently defined are:

```
#define F_RELFLG  0000001 /* relocation entries stripped */
```

```
#define F_EXEC     0000002 /* file is executable */
#define F_LNNO     0000004 /* line numbers stripped */
#define F_LSYMS    0000010 /* local symbols stripped */
#define F_MINMAL   0000020 /* minimal object file */
#define F_UPDATE   0000040 /* update file, ogen produced */
#define F_SWABD    0000100 /* file is "pre-swabbed" */
#define F_AR16WR   0000200 /* 16-bit DEC host */
#define F_AR32WR   0000400 /* 32-bit DEC host */
#define F_AR32W    0001000 /* non-DEC host */
#define F_PATCH    0002000 /* "patch" list in opt hdr */
#define F_80186    010000  /* contains 80186 instructions */
#define F_80286    020000  /* contains 80286 instructions */
#define F_BM32ID   0160000 /* WE32000 family ID field */
#define F_BM32B    0020000 /* file contains WE 32100 code */
#define F_BM32MAU  0040000 /* file reqs MAU to execute */
#define F_BM32RST  0010000 /* this object file contains restore
                              work around [3B15/3B2 only] */
```

See Also

time(S), fseek(S), a.out(F)

filesys

default information for mounting filesystems

Description

The **/etc/default/filesys** file contains information for mounting filesystems in the following format:

name=value [*name=value*] ...

value may contain white spaces if quoted, and newlines may be escaped with a backslash.

mnt(C) and *sysadmsh*(ADM) use the information in the **/etc/default/filesys** when the system comes up multiuser. The following names are defined for **/etc/default/filesys**:

bdev	Name of the block interface device.
cdev	Name of the character interface device.
size	Size in blocks.
mountdir	Directory on which the filesystem is mounted.
desc	A description of the filesystem. For example, "User filesystem."
mountflags	Any flags passed to the *mount*(ADM) command.
fsckflags	Any flags passed to the *fsck*(ADM) command.
rcmount	Whether or not to mount the filesystem when the system goes multiuser. Can be **yes, no**, or **prompt**. If set to **prompt**, you are prompted when it is time to mount the filesystem.

See Also

mount(ADM), mnt(C), sysadmsh(ADM)

filesystem

format of filesystem types

Syntax

#include <sys/fs/??filsys.h>
#include <sys/types.h>
#include <sys/param.h>

Description

Every filesystem storage volume (for example, a hard disk) has a common format for certain vital information. Every such volume is divided into a certain number of 1024-byte blocks. There are four fileystem types available:

S51K (UNIX fileystem)
XENIX
AFS (ACER Fast Filesystem)
DOS

The DOS filesystem is a 512-byte filesystem. (The DOS filesystem structure is shown in **/usr/include/fs/ dosfilsys.h**. This page does not discuss the format of the DOS filesystem in detail. Consult a DOS reference for more information.)

Block 0 is unused and is available to contain a bootstrap program or other information.

Block 1 is the *super-block*. The format of the S51K, AFS, and XENIX filesystem super-blocks are described in two files in the directory **/usr/include/fs**: **s5filsys.h** (S51K and AFS), **xxfilsys.h** (XENIX). The XENIX filesystem boot block is 1024-bytes; the S51K and UNIX boot blocks are 512-byte blocks. In these include files, *s_isize* is the address of the first data block after the i-list. The i-list starts just after the super-block in block 2; thus the i-list is *s_isize*-2 blocks long. *s_fsize* is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers. If an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the console. Moreover, the free array is cleared so as to prevent further allocation from a presumably corrupted free list.

The free list for S51K and XENIX volumes (but not AFS) is maintained as follows: The *s_free* array contains, in *s_free*[1], ..., *s_free*[*s_nfree*-1], up to NICFREE-1 numbers of free blocks. *s_free*[0] is the block number of the head of a chain of blocks constituting the free list. The first short in each free-chain block is the number (up to NICFREE) of free-block numbers listed in the next NICFREE longs of this chain member. The first of these NICFREE blocks is the link to the next member of the chain. To allocate a block: decrement *s_nfree*, and the new block is *s_free* [*s_nfree*]. If the new block number is 0, there are no blocks left, so give an error. If *s_nfree* becomes 0, read in the block named by the new block number, replace *s_nfree* by its first word, and copy the block numbers in the next NICFREE longs into the *s_free* array. To free a block, check if *s_nfree* is NICFREE; if so, copy *s_nfree* and the *s_free* array into it, write it out, and set *s_nfree* to 0. In any event set *s_free* [*s_nfree*] to the freed block's number and increment *s_nfree*.

In the AFS filesystem, the free list is maintained differently, The AFS freelist is organized as a bitmap, one bit per (1K) block in the filesystem. This organization makes it easy to find contiguous stretches of free blocks.

s_tfree is the total free blocks available in the filesystem.

s_ninode is the number of free i-numbers in the *s_inode* array. To allocate an inode: if *s_ninode* is greater than 0, decrement it and return *s_inode* [*s_ninode*]. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the *s_inode* array, then try again. To free an inode, provided *s_ninode* is less than NICINOD, place its number into *s_inode* [*s_ninode*] and increment *s_ninode*. If *s_ninode* is already NICINOD, do not bother to enter the freed inode into any table. This list of inodes only speeds up the allocation process. The information about whether the inode is really free is maintained in the inode itself.

s_tinode is the total free inodes available in the file system.

The following applies only to S51K and AFS fileystems: *s_state* indicates the state of the file system. A cleanly unmounted, not damaged file system is indicated by the FsOKAY state. After a file system has been mounted for update, the state changes to FsACTIVE. A special case is used for the root file system. If the root file system appears damaged at boot time, it is mounted but marked FsBAD. Lastly, after a file system has been unmounted, the state reverts to FsOKAY.

s_flock and *s_ilock* are flags maintained in the core copy of the filesystem while it is mounted and their values on disk are immaterial. The value of *s_fmod* on disk is also immaterial, and is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

s_ronly is a read-only flag to indicate write-protection.

s_time is the last time the super-block of the file system was changed, and is a double precision representation of the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the *s_time* of the super-block for the root file system is used to set the system's idea of the time.

I-numbers begin at 1, and the storage for inodes begins in block 2. Also, inodes are 64 bytes long, so 16 of them fit into a block. Therefore, inode *i* is located in block $(i+31)/16$, and begins $64\times((i+31) \pmod{16})$ bytes from its start. Inode 1 is reserved for future use. Inode 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each inode represents one file. For the format of an inode and its flags, see *inode*(F).

Files

/usr/include/sys/filsys.h

/usr/include/sys/stat.h

See Also

fsck(ADM), mkfs(ADM), inode(F)

fspec

format specification in text files

Description

It is sometimes convenient to maintain text files on the UNIX system with non-standard tabs, (i.e., tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets <: and :>. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns;

2. a - followed immediately by an integer *n*, indicating tabs at intervals of *n* columns;

3. a - followed by the name of a ‘‘canned’’ tab specification.

Standard tabs are specified by **t-8**, or equivalently, **t1,9,17,25**, etc. The canned tabs which are recognized are defined by the *tabs*(C) command.

s*size* The s parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t-8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

See Also

ed(C), newform(C), tabs(C)

gettydefs

speed and terminal settings used by getty

Description

The **/etc/gettydefs** file contains information used by *getty*(M) to set up the speed and terminal settings for a line. It supplies information on what the *login* prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a BREAK character.

Each entry in **/etc/gettydefs** has the following format:

> label# initial-flags # final-flags # login-prompt #next-label [# login-program]

Each entry must be followed by a carriage return and a blank line. The various fields can contain quoted characters of the form **\b**, **\n**, **\c**, etc., as well as *nnn*, where *nnn* is the octal value of the desired character. The various fields are:

label — This is the string against which *getty*(M) tries to match its second argument. It is often the speed, such as **1200**, at which the terminal is supposed to run, but it need not be (see below).

initial-flags — These flags are the initial *ioctl*(S) settings to which the terminal is to be set if a terminal type is not specified to *getty*(M). The flags that *getty*(M) understands are the same as the ones listed in **/usr/include/sys/termio.h** [see *termio*(M)]. Normally only the speed flag is required in the *initial-flags*. *getty*(M) automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until *getty*(M) executes *login*(M).

final-flags — Sets the same values as the *initial-flags*. These flags are set just prior to *getty* executing *login-program*. The speed flag is again required. The composite flag **SANE** is a composite flag that sets the following *termio*(M) parameters:

modes set:
CREAD BRKINT IGNPAR ISTRIP ICRNL IXON ISIG ICANON ECHO ECHOK OPOST ONLCR

modes cleared:
CLOCAL IGNBRK PARMRK INPCK INLCR IUCLC IXOFF XCASE ECHOE ECHONL NOFLSH OLCUC OCRNL ONOCR ONLRET OFILL OFDEL NLDLY CRDLY TABDLY BSDLY VTDLY FFDLY

The other two commonly specified *final-flags* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.

login-prompt Contains login prompt message that greets users. Unlike the above fields where white space is ignored (a space, tab, or new-line), it is included in the *login-prompt* field. The '@' in the login-prompt field is expanded to the first line (or second line if it exists) in **/etc/systemid** (unless the '@' is preceded by a '\'). Several character sequences are recognized, including:

\n	Linefeed
\r	Carriage return
\v	Vertical tab
nnn	(3 octal digits) Specify ASCII character
\t	Tab
\f	Form feed
\b	Backspace

next-label Identifies the next entry in *gettydefs* for *getty* to try if the current one is not successful. *Getty* tries the next label if a user presses the BREAK key while attempting to log in to the system. Groups of entries, for example, for dial-up lines or for TTY lines, should form a closed set so that *getty* cycles back to the original entry if none of the entries is successful. For instance, **2400** linked to **1200**, which in turn is linked to **300**, which finally is linked to **2400**.

login-program
The name of the program that actually logs the user in to Altos UNIX System V. The default program is **/etc/login**. If preceded by the keyword **AUTO**, *getty* will not prompt for a username, but instead uses its first argument as the username and executes the *login-program* immediately.

If *getty* is called without a second argument, then the first entry of **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. The first entry is also used if *getty* can not find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into the command which will bring up a terminal at **300** baud.

After modifying **/etc/gettydefs**, run it through *getty* with the check option to be sure there are no errors.

Files

/etc/gettydefs

See Also

stty(C), ioctl(S), getty(M), login(M)

gps

graphical primitive string, format of graphical files

Description

GPS is a format used to store graphical data. Several routines have been developed to edit and display GPS files on various devices. Also, higher level graphics programs such as *plot* [in *stat*(1G)] and *vtoc* [in *toc*(1G)] produce GPS format output files.

A GPS is composed of five types of graphical data or primitives.

GPS PRIMITIVES

lines The *lines* primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a *move* to that location. (A *move* is a relocation of the graphic cursor without drawing.) Successive points produce line segments from the previous point. Parameters are available to set *color*, *weight*, and *style* (see below).

arc The *arc* primitive has a variable number of points to which a curve is fit. The first point produces a *move* to that point. If only two points are included, a line connecting the points will result; if three points a circular arc through the points is drawn; and if more than three, lines connect the points. (In the future, a spline will be fit to the points if they number greater than three.) Parameters are available to set *color, weight,* and *style.*

text The *text* primitive draws characters. It requires a single point which locates the center of the first character to be drawn. Parameters are *color, font, textsize,* and *textangle.*

hardware The *hardware* primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the *hardware* string.

comment A *comment* is an integer string that is included in a GPS file but causes nothing to be displayed. All GPS files begin with a comment of zero length.

GPS PARAMETERS

color *Color* is an integer value set for *arc, lines,* and *text* primitives.

weight *Weight* is an integer value set for *arc* and *lines* primitives to indicate line thickness. The value **0** is narrow weight, **1** is bold, and **2** is medium weight.

style *Style* is an integer value set for *lines* and *arc* primitives to give one of the five different line styles that can be drawn on TEKTRONIX 4010 series storage tubes. They are:

- **0** solid
- **1** dotted
- **2** dot dashed
- **3** dashed
- **4** long dashed

font An integer value set for *text* primitives to designate the text font to be used in drawing a character string. (Currently *font* is expressed as a four-bit *weight* value followed by a four-bit *style* value.)

textsize *Textsize* is an integer value used in *text* primitives to express the size of the characters to be drawn. *Textsize* represents the height of characters in absolute *universe-units* and is stored at one-fifth this value in the size-orientation (*so*) word (see below).

textangle *Textangle* is a signed integer value used in *text* primitives to express rotation of the character string around the beginning point. *Textangle* is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (*so*) word as a value 256/360 of it's absolute value.

ORGANIZATION

GPS primitives are organized internally as follows:

lines	*cw points sw*
arc	*cw points sw*
text	*cw point sw so* [*string*]
hardware	*cw point* [*string*]
comment	*cw* [*string*]

cw *cw* is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word-count for that primitive.

point(s) *Point(s)* is one or more pairs of integer coordinates. *Text* and *hardware* primitives only require a single *point*. *Point(s)* are values within a Cartesian plane or *universe* having 64K (-32K to +32K) points on each axis.

sw *Sw* is the style-word and is used in *lines, arc,* and *text* primitives. For all three, eight bits contain *color* information. In *arc* and *lines* eight bits are divided as four bits *weight* and four bits *style*. In the *text* primitive eight bits of *sw* contain the *font*.

so *So* is the size-orientation word used in *text* primitives. Eight bits contain text size and eight bits contain text rotation.

string *String* is a null-terminated character string. If the string does not end on a word boundary, an additional null is added to the GPS file to insure word-boundary alignment.

See Also

graphics(ADM), stat(ADM), toc(ADM)

group

format of the group file

Description

group contains the following information for each group:

- Group name

- Numerical group ID

- Comma-separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a newline. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group IDs to names.

Files

/etc/group

See Also

newgrp(C), passwd(C), passwd(F)

Standards Conformance

group is conformant with:

The X/Open Portability Guide II of January 1987;
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

hs

High Sierra/ISO-9660 CD-ROM filesystem

Description

The hs filesystem module supports the mounting of CD-ROM filesystems conforming the High Sierra/ISO-9660 format.

Files

/usr/include/sys/fs/hs*

See Also

cdrom(HW), mkdev(ADM), mount(ADM)

Notes

Since the CD-ROM is a read-only device it is only possible to mount CD-ROM filesystems as read-only. The kernel enforces this regardless of whether the **-r** option of *mount*(ADM) was used when the filesystem was mounted.

The command *mkdev high-sierra* can be used to interactively configure High Sierra/ISO-9660 filesystem support.

inittab, init.base

script for the init process

Description

The **inittab** file supplies the script to *init*'s role as a general process dispatcher. The process that constitutes the majority of *init*'s process dispatching activities is the line process **/etc/getty** that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

The **inittab** file is recreated automatically by *idmkinit* at boot time anytime the kernel has been reconfigured. To construct a new **inittab** file, *idmkinit* reads the file **/etc/conf/cf.d/init.base**, which contains a base set of **inittab** entries that are required for the system, and combines these base entries with add-on entries from the device driver init files in the directory **/etc/conf/init.d**.

If you add an entry directly to **inittab**, the change exists only until the kernel is relinked. To add an entry permanently, you must also edit **/etc/conf/cf.d/init.base**. The **init.base** file has the same format as **inittab**.

The *inittab* file is composed of entries that are position-dependent and have the following format:

id:rstate:action:process

Each entry is delimited by a new-line; however, a backslash (\) preceding a new-line indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the *process* field using the *sh*(C) convention for comments. Comments for lines that spawn *getty*s are displayed by the *who*(C) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the **inittab** file. The entry fields are:

id This is up to four characters used to uniquely identify an entry.

rstate This defines the *run-level* in which this entry is to be processed. *Run-levels* effectively correspond to a configuration of processes in the system. That is, each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by a number ranging from **0** through **6**. As an example, if the system is in *run-level* **1**, only those entries having a **1** in the *rstate* field will be processed. When *init* is requested to change *run-levels,* all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (**SIGTERM**) and allowed a 20-second grace period before being forcibly terminated by a kill signal (**SIGKILL**). The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from **0-6**. If no *run-level* is specified, then the process is assumed to be valid at all *run-levels* **0-6**. There are three other values, **a**, **b**, and **c**, which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* [see *init*(M)] process requests them to be run (regardless of the current *run-level* of the system). They differ from *run-levels* in that *init* can never enter *run-level* **a**, **b**, or **c**. Also, a request for the execution of any of these processes does not change the current *run-level*. Furthermore, a process started by an **a**, **b**, or **c** command is not killed when *init* changes levels. They are only killed if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab**, or *init* goes into the *SINGLE USER* state.

action Key words in this field tell *init* how to treat the process specified in the *process* field. The actions recognized by *init* are as follows:

respawn If the process does not exist, then start the process, do not wait for its termination (continue scanning the **inittab** file); and when it dies, restart the process. If the process currently exists, then do nothing and continue scanning the **inittab** file.

wait Upon *init*'s entering the *run-level* that matches the entry's *rstate*, start the process and wait for its termination. All subsequent reads of the **inittab** file while *init* is in the same *run-level* will cause *init* to ignore this entry.

once Upon *init*'s entering a *run-level* that matches the entry's *rstate*, start the process, do not wait for its termination. When it dies, do not restart the process. If upon entering a new *run-level*, where the process is still running from a previous *run-level* change, the program will not be restarted.

boot The entry is to be processed only at *init*'s boot-time read of the **inittab** file. *init* is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match *init*'s *run-level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait The entry is to be processed the first time *init* goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process will run right after the boot.) *init* starts the process, waits for its termination and, when it dies, does not restart the process.

powerfail Execute the process associated with this entry only when *init* receives a power fail signal [**SIGPWR** see *signal*(S)].

powerwait Execute the process associated with this entry only when *init* receives a power fail signal (**SIGPWR**) and wait until it terminates before continuing any processing of **inittab**.

restart Execute the process associated with this entry only when *init* receives a power fail signal (**SIGPWR**) on a warm restart of the system after a power failure.

off If the process associated with this entry is currently running, send the warning signal (**SIGTERM**) and wait 20 seconds before forcibly terminating the process via the kill signal (**SIGKILL**). If the process is nonexistent, ignore the entry.

ondemand This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with *run-levels*. This is used only with the **a**, **b**, or **c** values described in the *rstate* field.

initdefault An entry with this *action* is only scanned when *init* initially invoked. *init* uses this entry, if it exists, to determine which *run-level* to enter initially. It does this by taking the highest *run-level* specified in the **rstate** field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **0123456** and so *init* will enter *run-level* **6**. Additionally, if *init* does not find an **initdefault** entry in **/etc/inittab**, then it will request an initial *run-level* from the user at reboot time.

sysinit Entries of this type are executed before *init* tries to access the console (i.e., before the **Console Login:** prompt). It is expected that this entry will be used only to initialize devices on which *init* might try to ask the *run-level* question. These entries are executed and waited for before continuing.

process This is a *sh* command to be executed. The entire **process** field is prefixed with *exec* and passed to a forked *sh* as **sh -c** **´exec** *command´*. For this reason, any legal *sh* syntax can appear in the *process* field. Comments can be inserted with the ; *#comment* syntax.

Files

/etc/inittab
/etc/conf/cf.d/init.base

See Also

idmkinit(ADM), sulogin(ADM), disable(C), enable(C), getty(M), init(M), powerfail(M), restart(M) sh(C), who(C), exec(S), open(S), signal(S)

inode

format of an inode

Syntax

#include <sys/types.h>
#include <sys/ino.h>

Description

An inode for a plain file or directory in a file system has the structure defined by **<sys/ino.h>**. For the meaning of the defined types *off_t* and *time_t* see *types* (F).

Files

/usr/include/sys/ino.h

See Also

stat(S), filesystem(F), types(F)

issue

issue identification file

Description

The file /etc/issue contains the *issue* or project identification to be printed as a login prompt. This is an ASCII file which is read by program *getty* and then written to any terminal spawned or respawned from the *lines* file.

Files

/etc/issue

See Also

login(M)

langinfo

language information constants

Syntax

#include <langinfo.h>

Description

This is a header file that contains constants used to identify items of *nl_langinfo*(S) data. (See *nl_langinfo*(S).)

The contents of the header file are shown below.

```
/*
 *      LC_CTYPE   is not queried through this interface
 */

/*
 *      LC_NUMERIC          items:
 */

#define RADIXCHAR           2000   /* Decimal separator */
#define THOUSEP             2001   /* Thousands separator */

/*
 *      LC_TIME   items:
*/
#define D_FMT               3000   /* Date format only */
#define T_FMT               3001   /* Time format only */
#define PM_STR              3002   /* PM */
#define AM_STR              3003   /* AM */
#define D_T_FMT             3004   /* Date and time format */
#define DAY_1               3005   /* Sunday */
#define DAY_2               3006   /* Monday */
#define DAY_3               3007   /* Tuesday */
#define DAY_4               3008   /* Wednesday */
#define DAY_5               3009   /* Thursday */
#define DAY_6               3010   /* Friday */
#define DAY_7               3011   /* Saturday */
#define ABDAY_1             3012   /* Sun */
#define ABDAY_2             3013   /* Mon */
#define ABDAY_3             3014   /* Tue */
#define ABDAY_4             3015   /* Wed */
#define ABDAY_5             3016   /* Thu */
#define ABDAY_6             3017   /* Fri */
```

```
#define ABDAY_7             3018    /* Sat */
#define MON_1               3019    /* January */
#define MON_2               3020    /* February */
#define MON_3               3021    /* March */
#define MON_4               3022    /* April */
#define MON_5               3023    /* May */
#define MON_6               3024    /* June */
#define MON_7               3025    /* July */
#define MON_8               3026    /* August */
#define MON_9               3027    /* September */
#define MON_10              3028    /* October */
#define MON_11              3029    /* November */
#define MON_12              3030    /* December */
#define ABMON_1             3031    /* Jan */
#define ABMON_2             3032    /* Feb */
#define ABMON_3             3033    /* Mar */
#define ABMON_4             3034    /* Apr */
#define ABMON_5             3035    /* May */
#define ABMON_6             3036    /* Jun */
#define ABMON_7             3037    /* Jul */
#define ABMON_8             3038    /* Aug */
#define ABMON_9             3039    /* Sep */
#define ABMON_10            3040    /* Oct */
#define ABMON_11            3041    /* Nov */
#define ABMON_12            3042    /* Dec */

/*
 *  LC_COLLATE              is not queried through this interface
 */

/*
 *  LC_MESSAGES             items:
 */

#define YESSTR      5000    /* Affirmative reply to y/n question */
#define NOSTR       5001    /* Negative reply to yes/no question */

/*
 *  LC_MONETARY                     items:
 */

#define CRNCYSTR            6000    /* Currency symbol */
```

See Also

nl_types(F), nl_langinfo(S)

Value Added

langinfo is an extension of AT&T System V provided in Altos UNIX System V.

ldfcn

common object file access routines

Syntax

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

Description

The common object file access routines are a collection of functions for reading common object files and archives containing common object files. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function *ldopen*(S) allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

LDFILE	*ldptr;
TYPE(ldptr)	The file magic number used to distinguish between archive members and simple object files.
IOPTR(ldptr)	The file pointer returned by *fopen* and used by the standard input/output functions.
OFFSET(ldptr)	The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.
HEADER(ldptr)	The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

(1) functions that open or close an object file

ldopen(S) and *ldaopen*[see *ldopen*(S)]
open a common object file
ldclose(S) and *ldaclose*[see *ldclose*(S)]
close a common object file

(2) functions that read header or symbol table information

ldahread(S)
read the archive header of a member of an archive file
ldfhread(S)
read the file header of a common object file
ldshread(S) and *ldnshread*[see *ldshread*(S)]
read a section header of a common object file
ldtbread(S)
read a symbol table entry of a common object file
ldgetname(S)
retrieve a symbol name from a symbol table entry or from the string table

(3) functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

ldohseek(S)
seek to the optional file header of a common object file
ldsseek(S) and *ldnsseek*[see *ldsseek*(S)]
seek to a section of a common object file
ldrseek(S) and *ldnrseek*[see *ldrseek*(S)]
seek to the relocation information for a section of a common object file
ldlseek(S) and *ldnlseek*[see *ldlseek*(S)]
seek to the line number information for a section of a common object file
ldtbseek(S)
seek to the symbol table of a common object file

(4) the function *ldtbindex*(S) which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages.

All the functions except *ldopen*(S), *ldgetname*(S), *ldtbindex*(S) return either **SUCCESS** or **FAILURE**, both constants defined in **ldfcn.h.** *Ldopen*(S) and *ldaopen*[(see *ldopen*(S)] both return pointers to an **LDFILE** structure.

Additional access to an object file is provided through a set of macros defined in **ldfcn.h.** These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the

LDFILE structure into a reference to its file descriptor field.

The following macros are provided:

```
GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)
```

The STROFFSET macro calculates the address of the string table. See the manual entries for the corresponding standard input/output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access routine library **libld.a**.

See Also

fseek(S), ldahread(S), ldclose(S), ldgetname(S), ldfhread(S), ldlread(S), ldlseek(S), ldohseek(S), ldopen(S), ldrseek(S), ldlseek(S), ldshread(S), ldtbindex(S), ldtbread(S), ldtbseek(S), stdio(S), intro(M)

Notes

The macro **FSEEK** defined in the header file **ldfcn.h** translates into a call to the standard input/output function *fseek*(S). **FSEEK** should not be used to seek from the end of an archive file, since the end of an archive file may not be the same as the end of one of its object file members.

limits

file header for implementation-specific constants

Syntax

#include <limits.h>

Description

The header file **<limits.h>** is a list of magnitude limitations imposed by a specific implementation of the operating system. All values are specified in decimal.

```
#define ARG_MAX     5120       /* max length of arguments to exec */
#define CHAR_BIT    8          /* # of bits in a "char" */
#define CHAR_MAX    127        /* max integer value of a "char" */
#define CHAR_MIN    -128       /* min integer value of a "char" */
#define CHILD_MAX   25         /* max # of processes per user id */
#define CLK_TCK     100        /* # of clock ticks per second */
#define DBL_DIG     16         /* digits of precision of a "double" */
#define DBL_MAX     1.79769313486231470e+308 /*max decimal value of a "double"*/
#define DBL_MIN     4.94065645841246544e-324 /*min decimal value of a "double"*/
#define FCHR_MAX    1048576    /* max size of a file in bytes */
#define FLT_DIG     7          /* digits of precision of a "float" */
#define FLT_MAX     3.40282346638528860e+38 /*max decimal value of a "float" */
#define FLT_MIN     1.40129846432481707e-45 /*min decimal value of a "float" */
#define HUGE_VAL    3.40282346638528860e+38 /*error value returned by Math lib*/
#define INT_MAX     2147483647 /* max decimal value of an "int" */
#define INT_MIN     -2147483648 /* min decimal value of an "int" */
#define LINK_MAX    1000       /* max # of links to a single file */
#define LONG_MAX    2147483647/* max decimal value of a "long" */
#define LONG_MIN    -2147483648         /* min decimal value of a "long" */
#define Name_MAX    14         /* max # of characters in a file name */
#define OPEN_MAX    60         /* max # of files a process can have open */
#define PASS_MAX    8          /* max # of characters in a password */
#define PATH_MAX    256        /* max # of characters in a path name */
#define PID_MAX     30000      /* max value for a process ID */
#define PIPE_BUF    5120       /* max # bytes atomic in write to a pipe */
#define PIPE_MAX    5120       /* max # bytes written to a pipe in a write */
#define SHRT_MAX    32767      /* max decimal value of a "short" */
#define SHRT_MIN    -32768     /* min decimal value of a "short" */
#define STD_BLK     1024       /* # bytes in a physical I/O block */
#define SYS_NMLN    9          /* # of chars in uname-returned strings */
#define UID_MAX     60000      /* max value for a user or group ID */
#define USI_MAX     4294967295/* max decimal value of an "unsigned" */
#define WORD_BIT    32         /* # of bits in a "word" or "int" */
```

Standards Conformance

limits is conformant with:

The X/Open Portability Guide II of January 1987.

linenum

line number entries in a common object file

Syntax

#include <linenum.h>

Description

The *cc* command generates an entry in the object file for each C source line on which a breakpoint is possible [when invoked with the **-g** option; see *cc*(CP)]. Users can then reference line numbers when using the appropriate software test system [see *sdb*(CP)]. The structure of these line number entries appears below.

```
struct  lineno
{
        union
        {
                long      l_symndx ;
                long      l_paddr ;
        }                 l_addr ;
        unsigned short    l_lnno ;
} ;
```

Numbering starts with one for each function. The initial line number entry for a function has *l_lnno* equal to zero, and the symbol table index of the function's entry is in *l_symndx*. Otherwise, *l_lnno* is non-zero, and *l_paddr* is the physical address of the code for the referenced line. Thus the overall structure is the following:

function symtab index	0
physical address	line
physical address	line
...	
function symtab index	0
physical address	line
physical address	line
...	
l_addr	*l_lnno*

See Also

cc(CP), sdb(CP), a.out(F)

logs

MMDF log files

Syntax

System status, error, and statistics logging for MMDF

Description

MMDF maintains run-time log files at several levels of activity. The primary distinction is among message-level, channel-level, and link-level information. All logging settings can be overridden by entries in the runtime tailor file . In MMDF, that member is merged with */usr/mmdf/log* to determine the full pathname to the log. Logs are protected so that any process may write into them, but only MMDF may read them (i.e., 0622).

The logging files may be the source of some confusion, since the llog package entails some complexity. Its three critical factors are coordinated access, restricted file length, and restricted verbosity.

The length of a logging file can be limited to 25-block units. This is extremely important since files can grow very long over a period of time, especially if there are many long messages sent or very verbose logging.

Restricted verbosity is a way of easily tuning the amount of text entered into the log. This is probably the one parameter you need most to worry about. Set to full tilt (level=FTR), MMDF get noticeably slower and I/O bound. It also does a pretty good job of showing what it is doing and hence helping you figure out the source of errors. When you get to trust the code, setting the logging level down is highly recommended. The lowest would be TMP or FAT, for temporary or fatal errors. GEN will log errors and general information. FST logs errors, general and statistics information.

Specific Logs

Even with the listed divisions, the logs contain a variety of information. Only the message-level log's format will be explained in significant detail.

msg.log
: records enqueue and dequeue transitions, by *submit* and *deliver*. Entries by a background *deliver* process are noted with a ''BG-xxxx'' tag, where the x's contain the 4 least-significant decimal digits of the daemon's process id. This is to allow distinguishing different daemons. When *deliver* is invoked, by *submit,* for an immediate attempt, the tag begins with ''DL'' rather than ''BG''. Entries by *submit* begin with ''SB''.

Every major entry will indicate the name of the message involved. Entries from *submit* will show ''lin'' if the submission is from a user on the local machine. In this case, the end of the entry will show the login name of the sender. If the entry is labelled ''rin,'' then the mail is being relayed. The channel name, source host, and sender address are shown. Within parentheses, the number of addressees and the byte-length of the message are listed.

Entries from *deliver* show final disposition of a message/addressee. These are indicated by ''end.'' Then, there is the destination channel and mailbox name. In brackets, the queue latency for the address is shown in hours, seconds, and minutes.

chan.log
: records activity by the channel programs, in chndfldir[]. Entries have a tag indicating the type of channel making the entry. Different channels record different sorts of information. For example, the *local* channel shows when a *rcvmail* private reception program is invoked.

ph.log
: is used by the telephone link-level (packet) code.

ph.trn
: is the one file that is *not* size-limited. It records a transcript of every character sent and received on a telephone channel. It is reset to zero length at the beginning of every phone session. It is kept verbose, in order to facilitate checking the status of any telephone channel which is active. Hence, just watching for the ph.trn file to get larger can indicate that there is progress. Each telephone channel may have its own transcript file specified in the channel definition in the runtime tailor file.

See Also

mmdf(ADM)

Value Added

logs is an extension of AT&T System V provided in Altos UNIX System V.

maildelivery

user delivery specification file

Description

The delivery of mail by the local channel can run through various courses, including using a user tailorable file. The delivery follows the following strategy, giving up at any point it considers the message delivered.

1) If the address indicates a pipe or file default, then that is carried out.

2) The file *.maildelivery* (or something similar) in the home directory is read if it exists and the actions in it are followed.

3) A system-wide file is consulted next, such as */usr/lib/maildelivery* and the actions are similar to step 2.

4) If the message still hasn't been delivered, then it is put into the user's normal mailbox (*.mail* or *mailbox*) depending on the system.

The format of the *.maildelivery* file is

field <*FS*> pattern <*FS*> action <*FS*> result <*FS*> string

where:

field is name of a field that is to be searched for a pattern. This is any header field that you might find in a message. The most commonly used headers are From, to, cc, subject and sender. As well as the standard headers, there are some pseudo-headers that can also be used. These are:

source The out-of-band sender information. This is the address MMDF would use for reporting delivery problems with the message.

addr the address that was used to mail to you, normally yourname or yourname=string (see below).

default if the message hasn't been delivered yet, this field is matched.

* this case is always true regardless of any other action.

pattern is some sequence of characters that may be matched in the above *field*. Case is not significant.

action is one of the mail delivery actions supported by the local channel. Currently the supported actions are **file** or **>**, which appends the message to the given file, with delimiters; **pipe** or **|**, which starts up a process with the message as the standard input; and **destroy** which throws the message away.
For piped commands, the exit status of the command is significant. An exit status of 0 implies that the command succeeded and everything went well. An exit status of octal 0300-0377 indicates that a permanent failure occurred and the message should be rejected. Any other exit status indicates a temporary failure and the delivery attempt will be aborted and restarted at a later time.

result is one of the letters A, R or ? which stand for Accept, Reject and "Accept if not delivered yet". They have the following effects:

A If the result of this line's action is OK, then the message can be considered delivered.

R The message is not to be considered delivered by this action.

? This is equivalent to *A* except that the action is not carried out if the message has already been accepted.

The file is always read completely so that several matches can be made, and several actions taken. As a security check, the *.maildelivery* file must be owned by either the user or root, and must not have group or general write permission. In addition the system delivery file has the above restrictions but must also be owned by root. If the field specified does not need a pattern, a dash (-), or similar symbol is usually inserted to show that the field is present but not used. The field separator character can be either a tab, space or comma (,). These characters can be included in a string by quoting them with double quotes ("); double quotes can be included with a backslash '\'.

MMDF treats local addresses which contain an equal sign ('=') in a special manner. Everything in a local address from an equals sign to the '@' is ignored and passed on to the local channel. The local channel will make the entire string available for matching against the *addr* string of the *.maildelivery* file. For example, if you were to subscribe to a digest as "foo=digest@bar.NET", *submit*(ADM) and the local channel will verify that it is legal to deliver to "foo", but then the entire string "foo=digest" will be available for string matching against the *.maildelivery* file for the **addr** field.

Environment

The environment in which piped programs are run contains a few standard features, specifically:

HOME is set to the user's home directory.
USER is set to the user's login name.
SHELL is set to the user's login shell (defaults to **/bin/sh**).

The default umask is set up to 077, this gives a very protective creation mask. If further requirements are needed, then a shell script can be run first to set up more complex environments.

There are certain built-in variables that you can give to a piped program. These are *$(sender)*, *$(address)*, *$(size)*, *$(reply-to)* and *$(info)*. *$(sender)* is set to the return address for the message. *$(address)* is set to the address that was used to mail to you, normally 'yourname' or 'yourname=string'. *$(size)* is set to the size in bytes of this message. *$(reply-to)* is set to the Reply-To: field (or the From: field if the former is missing) and so can be used for automatic replies. *$(info)* is the info field from the internal mail header and is probably only of interest to the system maintainers.

Example

Here is a rough idea of what a *.maildelivery* file looks like:

```
# lines starting with a '#' are ignored.
# as are blank lines
# file mail with mmdf2 in the "To:" line into file mmdf2.log
To      mmdf2     file     A     mmdf2.log
# Messages from mmdf pipe to the program err-message-archive
From     mmdf     pipe     A     err-message-archive
# Anything with the "Sender:" address "uk-mmdf-workers"
# file in mmdf2.log if not filed already
Sender    uk-mmdf-workers     file     ?     mmdf2.log
# "To:" unix - put in file unix-news
To     Unix     >     A     unix-news
# if the address is jpo=mmdf - pipe into mmdf-redist
Addr     jpo=mmdf     |     A     mmdf-redist
# if the address is jpo=ack - send an acknowledgement copy back
Addr     jpo=ack     |     R     resend  -r  $(reply-to)
# anything from steve - destroy!
from     steve     destroy     A     -
# anything not matched yet - put into mailbox
default     -     >     ?     mailbox
# always run rcvalert
```

```
*     -     |     R     rcvalert
```

Files

$HOME/.maildelivery	normal location
/usr/lib/maildelivery	the system file. This should be protected.

The **/usr/lib/maildelivery** file contains contents such as:

```
default     -     pipe     A     stdreceive
*           -     pipe     R     ttynotify
```

This allows the system to interfaces with non-standard mail systems that do not support delimiter-separated mailboxes.

See Also

rcvtrip(C)

mapchan

format of tty device mapping files

Description

mapchan configures the mapping of information input and output.

Each unique *channel* map requires a multiple of 1024 bytes (a 1K buffer) for mapping the input and output of characters. No buffers are required if no *channels* are mapped. If control sequences are specified, an additional 1K buffer is required.

A method of sharing maps is implemented for *channels* that have the same map in place. Each additional, unique map allocates an additional buffer. The maximum number of map buffers available on a system is configured in the kernel, and is adjustable via the link kit **NEMAP** parameter (see *configure* (ADM)). Buffers of maps no longer in use are returned for use by other maps.

Example of a Map File

The internal character set is defined by the right column of the input map, and the first column of the output map in place on that line. The default internal character set is the 8-bit ISO 8859/1 character set, which is also known as dpANS X3.4.2 and ISO/TC97/SC2. It supports the Latin alphabet and can represent most European languages.

Any character value not given is assumed to be a straight mapping, only the differences are shown in the *mapfile*. The left hand columns must be unique. More than one occurrence of any entry is an error. Right hand column characters can appear more than once. This is "many to one" mapping. Nulls can be produced with compose sequences or as part of an output string.

It is recommended that no mapping be enabled on the *channel* used to create or modify the mapping files. This prevents any confusion of the actual values being entered due to mapping. It is also recommended that numeric rather than character representations be used in most cases, as these are not likely to be subject to mapping. Use comments to identify the characters represented. Refer to the *ascii* (M) manual page and the hardware reference manual for the device being mapped for the values to assign.

```
#
# sharp/pound/cross-hatch is the comment character
# however, a quoted # ('#') is 0x23, not a comment
#
# beep, input, output, dead, compose and
# control are special keywords and should appear as shown.
#

    beep            # sound the bell when errors occur
    input

    a b
    c d

    dead p
    q r             # p followed by q yields r.
    s t             # p followed by s yields t.

    dead u
    v w             # u followed by v yields w.

    compose x       # x is the compose key (only one allowed).
    y z A
    B C D           # x followed by B and C yields D.

    output
    e f             # e is mapped to f.
    g h i j         # g is mapped to hij - one to many.
    k l m n o       # k is mapped to lmno.

    control         # The control sections must be last

    input
    E  1            # The character E is followed by 1 more
                    unmapped character

    output
    FG  2           # The characters FG are followed by 2
                    more unmapped characters
```

All of the single letters above preceding the ''control'' section must be in one of these formats:

56	# decimal
045	# octal
0xfa	# hexadecimal
'b'	# quoted char
'\076'	# quoted octal
'\x4a'	# quoted hex

All of the above formats are translated to single byte values.

The **control** sections (which must be the last in the file) contain specifications of character sequences which should be passed through to or from the terminal device without going through the normal *mapchan* processing. These specifications consist of two parts: a fixed sequence of one or more defined characters indicating the start of a no-map sequence, followed by a number of characters of which the actual values are unspecified.

To illustrate this, consider a cursor-control sequence which should be passed directly to the terminal without being mapped. Such a sequence would typically begin with a fixed escape sequence instructing the terminal to interpret the following two characters as a cursor position; the values of the following two characters are variable, and depend on the cursor position requested. Such a control sequence would be specified as:

```
\E=  2                  # Cursor control: escape = <x> <y>
```

There are two subsections under **control**: the **input** section is used to filter data sent from the terminal to Altos UNIX System V, and the **output** section is used to filter data sent from Altos UNIX System V to the terminal. The two fields in each control sequence are separated by white space, that is the SPACE or TAB characters. Also the '#' (HASH) character introduces a comment, causing the remainder of the line to be ignored. Therefore, if any of these three characters are required in the specification itself, they should be entered using one of alternative means of entering characters, as follows:

^*x* The character produced by the terminal on pressing the CONTROL and *x* keys together.

\E or \e
The ESCAPE character, octal 033.

c Where *c* is one of b, f, l, n, r or t, produces BACKSPACE , FORM FEED , LINE FEED , NEWLINE , CARRIAGE RETURN or TAB characters respectively.

\0 Since the NULL character can not be represented, this sequence is stored as the character with octal value 0200, which behaves as a NULL on most terminals.

nn or *nnn*
Specifies the octal value of the character directly.

\ followed by any other character is interpreted as that character. This can be used to enter SPACE , TAB , or HASH characters.

Diagnostics

mapchan performs these error checks when processing the mapfile:

- More than one compose key.
- Characters mapped to more than one thing.
- Syntax errors in the byte values.
- Missing input or output keywords.
- Dead or compose keys also occurring in the input section.
- Extra information on a line.
- Mapping a character to null.
- Starting an output control sequence with a character that is already mapped.

If characters are displayed as the 7-bit value instead of the 8-bit value, use **stty -a** to verify that **-istrip** is set. Make sure **input** is mapping to the 8859 character set, **output** is mapping from the 8859 to the device display character set. **dead** and **compose** sequences are **input** mapping and should be going to 8859.

Files

/etc/default/mapchan
/usr/lib/mapchan/*

See Also

ascii(M), keyboard(HW), lp(C), lpadmin(ADM), mapchan(M), trchan(M), mapkey(M), parallel(HW), screen(HW), serial(HW), setkey(M), tty(M)

Notes

Some non-U.S. keyboards and display devices do not support characters commonly used by UNIX command shells and the C programming language. Do not attempt to use such devices for system administration tasks.

Not all terminals or printers can display all the characters that can be represented using this utility. Refer to the device's hardware manual for information on the capabilities of the peripheral device.

Warnings

Use of mapping files that specify a different "internal" character set per-channel, or a set other than the 8-bit ISO 8859 set supplied by default can cause strange side effects. It is especially important to retain the 7-bit ASCII portion of the character set (see *ascii*(M)). Altos UNIX System V utilities and applications assume these values. Media transported between machines with different internal code set mappings may not be portable as no mapping is performed on block devices, such as tape and floppy drives. *trchan* can be used to "translate" from one internal character set to another.

Do not set ISTRIP (see *stty*(C)) on channels that have mapping that includes eight bit characters.

Value Added

mapchan is an extension of AT&T System V provided in Altos UNIX System V.

maxuuscheds

UUCP uusched(ADM) limit file

Description

The **Maxuuscheds** **(/usr/lib/uucp/Maxuuscheds)** file contains a numerical string to limit the number of simultaneous **uusched** programs running. Each **uusched** running will have one **uucico** associated with it; limiting the number will directly affect the load on the system. The limit should be less than the number of outgoing lines used by UUCP (a smaller number is often desirable). This file is delivered with a default entry of 2. Again, this may be changed to meet the needs of the local system. However, keep in mind that the load on the system increases with the number of **uusched** programs running.

See Also

uucico(ADM), uucp(C), uusched(ADM), uux(C), uuxqt(C)

maxuuxqts

UUCP uuxqt(C) limit file

Description

The **Maxuuxqts** **(/usr/lib/uucp/Maxuuxqts)** file contains an ASCII number to limit the number of simultaneous **uuxqt** programs running. This file has a default entry of 2. If there is a lot of traffic from **mail**, you can increase this number to reduce the time it takes for the mail to leave your system. Keep in mind that the load on the system increases with the number of **uuxqt** programs running.

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C)

mdevice

device driver module description file

Syntax

/etc/conf/cf.d/mdevice

Description

The **mdevice** file is included in the directory **/etc/conf/cf.d**. It includes a one-line description of each device driver and configurable software module in the system to be built [except for file system types, see *mfsys*(F)]. Each line in **mdevice** represents the **Master** file component from a Driver Software Package (DSP) either delivered with the base system or installed later via *idinstall*(ADM).

Each line contains several whitespace-separated fields; they are described below. Each field must be supplied with a value or a '-' (dash).

1. *Device name*: This field is the internal name of the device or module, and may be up to 8 characters long. The first character of the name must be an alphabetic character; the others may be letters, digits, or underscores.

2. *Function list*: This field is a string of characters that identify driver functions that are present. Using one of the characters below requires the driver to have an entry point (function) of the type indicated. If no functions in the following list are supplied, the field should contain a dash.

   o - open routine

   c - close routine

   r - read routine

   w - write routine

   i - ioctl routine

   s - startup routine

   I - init routine

h - halt routine

p - poll routine

E - enter routine

P - pre-main init routine

f - fork routine

e - exec routine

x - exit routine

X - kexit routine

S - switch routine

U - restart routine

D - shut routine

Note that if the device is a 'block' type device (see field 3. below), a *strategy* routine and a *print* routine are required by default.

3. *Characteristics of driver*: This field contains a set of characters that indicate the characteristics of the driver. If none of the characters below apply, the field should contain a dash. The legal characters for this field are:

i - The device driver is installable.

n - The device driver is not installable.

a - The device driver is automatically installed.

c - The device is a 'character' device.

b - The device is a 'block' device.

t - The device is a tty.

o - This device may have only one *sdevice* entry.

r - This device is required in all configurations of the Kernel. This option is intended for drivers delivered with the base system only. Device nodes (special files in the */dev* directory), once made for this device, are never removed. See *idmknod.*

S - This device driver is a STREAMS module.

H - This device driver controls hardware. This option distinguishes drivers that support hardware from those that are entirely software (pseudo-devices).

G - This device does not use an interrupt though an interrupt is specified in the *sdevice* entry. This is used when you wish to associate a device to a specific device group.

D - This option indicates that the device driver can share its DMA channel.

O - This option indicates that the IOA range of this device may overlap that of another device.

s - Supress device count field.

N - This device has no driver.

R - This driver has a reset routine.

C - This driver allows cluster I/O requests (block device only).

Z - This device may have multiple *mdevice*(F) entries.

I - This driver ignores **pack.d** entries.

4. *Handler prefix*: This field contains the character string prepended to all the externally-known handler routines associated with this driver. The string may be up to 4 characters long.

5. *Block Major number*: This field should be set to zero in a DSP **Master** file. If the device is a 'block' type device, a value will be assigned by *idinstall* during installation.

6. *Character Major number*: This field should be set to zero in a DSP *Master* file. If the device is a 'character' type device (or 'STREAMS' type), a value will be assigned by *idinstall* during installation.

7. *Minimum units*: This field is an integer specifying the minimum number of these devices that can be specified in the **sdevice** file.

8. *Maximum units*: This field specifies the maximum number of these devices that may be specified in the **sdevice** file. It contains an integer.

9. *DMA channel*: This field contains an integer that specifies the DMA channel to be used by this device. If the device does not use DMA, place a '-1' in this field. Note that more than one de-

vice can share a DMA channel (previously disallowed).

Specifying STREAMS Devices and Modules

STREAMS modules and drivers are treated in a slightly different way from other drivers in all UNIX systems, and their configuration reflects this difference. To specify a STREAMS device driver, its **mdevice** entry should contain both an 'S' and a 'c' in the *characteristics* field (see 3. above). This indicates that it is a STREAMS driver and that it requires an entry in the UNIX kernel's *cdevsw* table, where STREAMS drivers are normally configured into the system.

A STREAMS module that is not a device driver, such as a line discipline module, requires an 'S' in the *characteristics* field of its **mdevice** file entry, but should not include a 'c', as a device driver does.

See Also

mfsys(F), sdevice(F), idinstall(ADM)

meisa

master EISA system kernel configuration file

Description

The **meisa** file is used by *uconfig*(ADM) to maintain a mapping between EISA hardware functions and UNIX device drivers. Each line in the **meisa** file specifies one such mapping.

Each line must contain the following five fields, each separated by a whitespace:

dev_name flags EISA_ID EISA_type majors

Each field is described below:

dev_name This field contains the internal name of the driver. This name must match the names in the first field of an **mdevice** file entry. If this field contains a single dash character (-), then the EISA function specified later in this line (in *func_type*) will not be mapped to a device driver (*uconfig* will ignore this EISA function).

flags This field is a string of characters that identify some special driver characteristics. If no flags are specified, the field should contain a single dash character (-). The legal flag characters are listed below:

f Indicates that this driver contains an EISA NMI (non-maskable interrupt) failure-handling routine.

i Indicates that there is an associated shell script that sets up the **/etc/conf/init.d/***dev_name* file. This shell script will be invoked by *uconfig* with the appropriate arguments.

n Indicates that there is an associated shell script that sets up the **/etc/conf/node.d/***dev_name* file. This shell script will be invoked by *uconfig* with the appropriate arguments.

EISA_ID This field contains the EISA board ID string that is associated with the driver. A full seven-character ID (for example, ACS0201) can be used, but the minor revision number may be omitted (for example, ACS02 or ACS020). This field is mandatory.

EISA_type This field contains the EISA function type string that is associated with the driver. This string can be a simple EISA type (for example, COM,MDC), or may include subtype information (such as COM,ASY;COM1). In addition, an abbreviated type string can be used. For example, COM,ASY will match both COM,ASY;COM1 and COM,ASY;COM2 on the specified board. If a type string is not defined for the EISA function, the following syntax may be used:

[*func_no*]

where *func_no* is the function number of the particular EISA board. For example, [5] specifies function 5. For a pseudo-driver that does not directly drive a hardware function, this field should be a single dash character (-).

majors This field normally should be 0 (zero), except for drivers with multiple major device numbers. In a such a case, this field should contain the number of major numbers occupied per instance of the hardware function.

Files

/etc/conf/cf.d/meisa

See Also

uconfig(ADM), mdevice(F), sdevice(F)

Notes

No more than one NMI failure handling routine may be specified for functions in a single EISA slot.

Value Added

meisa is an extension to AT&T UNIX System V provided in Altos UNIX System V.

mem, kmem

memory image file

Description

The **mem** file provides access to the computer's physical memory. All byte addresses in the file are interpreted as memory addresses. Thus, memory locations can be examined in the same way as individual bytes in a file. Note that accessing a nonexistent location causes an error.

The **kmem** file is the same as **mem** except that it corresponds to kernel virtual memory rather than physical memory.

In rare cases, the **mem** and **kmem** files may be used to write to memory and memory-mapped devices. Such patching is not intended for the naive user and may lead to a system crash if not conducted properly. Patching device registers is likely to lead to unexpected results if the device has read-only or write-only bits.

Files

/dev/mem

/dev/kmem

memtune

table of tunable parameters to be adjusted when adding more memory

Description

The **memtune** file contains a table of tunable parameters and rules on how they should be increased when more memory is added. It is used by *idmemtune*(ADM) to achieve a more efficiently tuned system.

Each line in **memtune** must contain the following four fields, each separated by whitespace:

param base increm max

Each field is described below:

param This field contains the name of the tunable parameter.

base This field specifies the value that *idememtune*(ADM) uses to set the base memory configuration. Base memory, by default, is 8 megabytes.

increm This field contains the value added to *base* for each 4-megabyte increase in system memory above the base memory configuration.

max This field specifies the maximum value that the tunable parameter *param* will be set by *idmemtune*(ADM).

Files

/etc/conf/cf.d/memtune

See Also

idmemtune(ADM), stune(F), uconfig(ADM)

Value Added

memtune is an extension to AT&T System V provided in Altos UNIX System V.

mfsys

configuration file for filesystem types

Syntax

/etc/conf/cf.d/mfsys

Description

The **mfsys** file contains configuration information for filesystem types that are to be included in the next system kernel to be built. It is included in the directory **/etc/conf/cf.d**, and includes a one-line description of each filesystem type. The **mfsys** file is gathered from component files in the directory **/etc/conf/mfsys.d.** Each line contains the following whitespace-separated fields:

1. *name*: This field contains the internal name for the filesystem type (example: S51K, AFS). This name is no more than 32 characters long, and by convention is composed of uppercase alphanumeric characters.

2. *prefix*: The *prefix* in this field is the string prepended to the *fstypsw* handler functions defined for this filesystem type (example: s5) The prefix must be no more that 8 characters long.

3. *flags*: The *flags* field contains a hex number of the form ''0xNN'' to be used in populating the *fsinfo* data structure table entry for this filesystem type.

4. *notify flags*: The *notify flags* field contains a hex number of the form ''0xNN'' to be used in population the *fsinfo* data structure table entry for this filesystem type.

5. *function bitstring*: The *function bitstring* is a string of 28 zeros and ones. Each filesystem type potentially defines 28 functions to populate the *fstypsw* data structure table entry for itself. All filesystem types do not supply all the functions in this table, however, and this bitstring is used to indicate which of the functions are present and which are absent. A ''1'' in this string indicates that a function has been supplied, and a ''0'' indicates that a function has not been supplied. Successive characters in the string represent successive elements of the *fstypsw* data structure, with the first entry in this data structure represented by the rightmost character in the string.

See Also

sfsys(F), idinstall(ADM), idbuild(ADM)

micnet

the Micnet default commands file

Syntax

/etc/default/micnet

Description

The **micnet** file lists the system commands that may be executed through the *remote* command. The file is required for each system in a Micnet network. Whenever a *remote* command is received through the network, the Micnet programs search the **micnet** file for the system command specified with the *remote* command. If found, the command is executed. Otherwise, the command is ignored and an error message is returned to the system which issued the *remote* command.

The file may contain one or more lines. If all commands may be executed, only the line

executeall

is required in the file. Otherwise, the commands must be listed individually. A line that defines an individual command has the form:

command=commandpath

Command is the command name to be specified in a *remote* command. **Commandpath** is the full pathname of the command on the specified system. The equal sign (=) separates the command and commandpath. For example, the line:

cat=/bin/cat

defines the command name *cat* (used in the *remote* command) to refer to the system command *cat* in the **/bin** directory.

When *executeall* is set, commands are sought in a series of default directories. Initially, the directories are **/bin** and **/usr/bin**. The default directories can be explicitly defined in the file by including a line of the form:

execpath=PATH=directory[:directory]...

The first part of the line, *execpath=PATH=*, is required. Each **directory** must be a valid pathname. The colon is required to separate directories. For example, the line:

execpath=PATH=/bin:/usr/bin:/usr/bobf/bin

sets the default directories to **/bin, /usr/bin,** and **/usr/bobf/bin.**

Files

/etc/default/micnet

See Also

aliases(M), netutil(ADM), systemid(F), top(F)

Notes

The **rcp** command cannot be executed from a remote system unless the **micnet** file contains either *executeall*, or the line

rcp=/usr/bin/rcp

Value Added

micnet is an extension of AT&T System V provided in Altos UNIX System V.

mmdftailor

provides run-time tailoring for the MMDF mail router

Description

The MMDF mail router reads site-dependent information from the ASCII file **/usr/mmdf/mmdftailor** each time it starts up.

Keywords in the tailor file are not case-sensitive; however, case is important for filenames and similar values. Use quotation marks to delimit strings to prevent them from being parsed into separate words accidentally.

The following alphabetical list describes most of the information you can set in the **mmdftailor** file. For information about additional channel-specific settings, refer to the documentation about the particular channel.

ALIAS defines an alias table. The following parameters can be used:

table	specifies the name of the table to be associated with this alias entry
trusted	allows the entries in the table to cause delivery to files and pipes
nobypass	does not allow the *~address* alias bypass mechanism to work on this file

Here is an example:

```
ALIAS table=sysaliases, trusted, nobypass
```

AUTHLOG controls authorization information. See MCHANLOG and MLOGDIR.

AUTHREQUEST is the address to which users should mail if they have questions about why a message was not authorized for delivery. It is also used as the sender of authorization warning messages. It is not used if authorization is not enabled on some channel. See the *auth* parameter under MCHN.

MADDID controls whether *submit* adds *Message-ID:* header lines if they are missing from messages. It takes a number as an argument. If the number is 0, no action is taken. If the number is non-zero, then *submit* adds *Message-ID:* header lines if they are missing from messages.

MADDRQ is the address files directory. If it is not a full pathname, it is taken relative to MQUEDIR.

MCHANLOG controls MMDF logging, except for authorization information and information produced by *deliver* and *submit*. See also MMSGLOG, AUTHLOG, and MLOGDIR.

Logging files and levels can also be specified in the channel descriptions. The logging file, if specified there, overrides the MCHANLOG definition. The logging level for the channel is set to the maximum of the MCHANLOG level and the channel description's level. The MCHANLOG level can therefore be used to increase logging on all channels at once.

Here is an example:

```
MCHANLOG /tmp/mmdfchan.log, level=FST, size=40,
         stat=SOME
```

An argument without an equal sign is taken as the name of the log. Logging levels are:

FAT	logs fatal errors only
TMP	logs temporary errors and fatal errors
GEN	saves the generally interesting diagnostics
BST	shows some basic statistics
FST	gives full statistics
PTR	shows a program trace listing of what is happening
BTR	shows more detailed tracing
FTR	saves every possible diagnostic

The BTR and FTR conditions are enabled only if you have compiled the MMDF system with DEBUG #define'd. This amount of tracing can quickly fill up log files. During normal operation, the logging level should be set somewhere between GEN and FST.

The *size* parameter is the number of 25-block units you will allow your log file to grow to. When a log file reaches that size, that logging either stops or cycles around overwriting old data (see **cycle**).

The *stat* parameter sets up various status flags for logging:

close	closes the log after each entry; this allows other processes to write to it as well
wait	if the log is busy, waits a while for it to free
cycle	if the log is at the maximum length specified with the *size* parameter, then cycles to the beginning

some — sets the values **close** and **wait** (the most common setting)

timed — opens the log and, after the timeout period (e.g., 5 minutes), closes the log and reopens it; this option overrides all other options (used to reduce the overhead of re-opening the log for every entry while still retaining the ability to move the log file out from under a running process and have the process begin logging in the new log file soon thereafter)

Tailoring of the log files is generally done at the end of the tailor file to prevent logging the tailoring action itself, thereby needlessly filling the log files when higher tracing levels are enabled. If you have bugs in the tailoring, you can move the log-file tailoring closer to the top of the tailor file.

MCHN defines a channel. The following parameters can be used:

name — the name of the channel

show — a display line, which is used for pretty printing purposes to explain what the channel is all about

que — the queue directory into which messages for this channel should be queued

tbl — the associated table entry of hosts that are accessible on this channel; if the specified table has not been previously defined, it will be defined with the same *name*, *file*, and *show* parameters as for this channel (do not define two channels that process the same queue, but use different tables because it will cause queue structure problems)

pgm — the channel program to invoke for this channel

mod — the mode in which this channel works; if several values are selected, they are cumulative:

- reg — regular mode (the default); the channel must be explicitly invoked by running *deliver*

- host — same as **reg**, but causes *deliver* to sort by host after sorting by channel, which allows as many mail messages as possible to get sent to a particular host before the connection is broken

bak	channel can be invoked only by the background deliver daemon
psv	channel is passive; it is a pickup-type channel (e.g., pobox)
imm	channel can be invoked immediately; no need to batch up mail
pick	channel can pick up mail from the remote host
send	channel can send mail to the remote host

ap — the address-parser type to be used for reformatting headers on messages going out on this channel; if several values are selected, they are cumulative:

same	does not reformat headers
822	selects RFC822-style source routes (e.g., @A:B@C)
733	selects %-style (JNT) source routes (e.g., B%C@A)
big	selects NRS hierarchy ordering (for the UK)
nodots	selects output of leftmost part of domain names (e.g., A in A.B.C) for sites that cannot handle domains (usually used in conjunction with the *known=* parameter to hide domain names behind a smart relay)
jnt	is equivalent to **733** combined with **big**

lname — a name overriding the default MLNAME value for this channel (used when the channel should have non-standard values for the local domain)

ldomain — a name overriding the default MLDOMAIN value for this channel host — the name of the host that is being contacted by this channel, usually used in the phone and pobox channels, or the name of the relay host when all mail to hosts in this channel's table gets relayed to one host (this is required on the *badusers* and *badhosts* pseudo-channels; it must be set to the local host for the list channel)

poll — the frequency of polling the remote machine (0 disables polling, -1 requests polling to be done every time the channel is started up, any other value is the number of 15-minute intervals to wait between polls); if any mail is waiting to be sent, this value is ignored because a connection is always attempted

insrc — a table of hosts controlling message flow

outsrc — see *insrc*

indest — see *insrc*

outdest — see *insrc*

known — a table of hosts that are known on this channel; be sure that the table contains your own fully specified host name

confstr — a string used by some channels for specifying the invocation of another program

auth — specifies the authorization tests that are made on this channel:

- free — default, no checking takes place
- inlog — log information for incoming messages
- outlog — log information for outgoing messages
- inwarn — warn sender of incoming message if authorization is inadequate (the message still goes through)
- outwarn — as **inwarn**, but for outgoing messages
- inblock — reject incoming messages that have inadequate authorization

outblock	as **inblock**, but for outgoing messages
hau	host and user authorizations are required
dho	(direct host only) when applying host controls, the message must be associated with a user local to that host (i.e., no source routes)

ttl (time-to-live) specifies the number of minutes for which retries to a host are blocked when *deliver* detects a connection failure to that host; this value can be overridden on the *deliver* command line (default is 2 hours)

log the name of the channel log file to be used instead of the default MCHANLOG

level the logging level for this channel (see also MCHANLOG)

Here is a simple example:

```
MCHN name=local, que=local, tbl=local,
     show="Local Delivery", pgm=local,
     poll=0, mod=imm, ap=822, level=BST
```

If the first argument does not have an equal sign, the values of the *name*, *que*, *tbl*, *pgm*, and *show* parameters take on this value.

MCHNDIR is where the channel programs are to be found.

MCMDDIR is the default commands directory where the various MMDF commands are located. Any command that does not have a full pathname is taken relative to this directory.

MDBM tells MMDF where to find the database file containing the associative store. DBM-style databases get their speed and flexibility by dynamic hashing and can get quite large. By default, the file is located in the MTBLDIR directory, but it might need to be relocated due to its size.

MDFLCHAN sets the default channel to something other than local.

MDLV is the name of the file used for tailoring the delivery for each user.

MDLVRDIR is the directory in which to deliver mail. If this is null, then the user's home directory is used. See also MMBXNAME and MMBXPROT.

MDMN defines a domain. The following parameters can be used:

name — an abbreviated name for the domain

show — a display line, which is used for pretty printing purposes to explain what the domain is all about

dmn — the full name (x.y.z...) of this domain

table — the associated table entry of known sites in this domain; if the specified table has not been previously defined, it will be defined with the same *name*, *file*, and *show* parameters as for this domain

Here is an example:

```
MDMN name="Root", dmn="", show="Root Domain",
     table=rootdomain
```

If the first argument does not have an equal sign, the values of the *name*, *dmn*, and *show* parameters take on this value. If no *table* parameter is specified, it defaults to the value of the name parameter.

MFAILTIME is the time a message can remain in a queue before a failed-mail message is sent to the sender and the message is purged from the queue. See also MWARNTIME.

MLCKDIR is the directory where the locking of files takes place, this is dependent on what style of locking you are doing.

MLDOMAIN gives your full local domain (this, combined with the MLNAME, and possibly the MLOCMACHINE, gives the full network address).

MLISTSIZE specifies the maximum number of addresses in a message before it is considered to have a ''big'' list. If there are more than the maximum number of addresses, then MMDF does not send a warning message for waiting mail and only returns a ''citation'' for failed mail. A citation consists of the entire header plus the first few lines of the message body.

MLNAME is the name of your machine or site as you wish it to be known throughout the network, which can be a generic host name used to hide a number of local hosts. If it is a generic host name, internal hosts are differentiated by MLOCMACHINE. See also MLDOMAIN.

MLOCMACHINE is the local name of the machine. It is used by a site that has several machines, but wants the machines themselves to appear as one address. See also MLNAME and MLDOMAIN.

MLOGDIR is the default directory in which the log files are kept. See also MMSGLOG, AUTHLOG, and MCHANLOG.

MLOGIN is the user who owns the MMDF transport system.

MMAXHOPS specifies the maximum number of *Received:* or *Via:* lines in a message before it is considered to be looping and is rejected.

MMAXSORT controls sorting of messages based on the number of messages in the queue. If the number of messages in the queue is less then MMAXSORT, the messages are sorted by arrival time and are dispatched in that order; otherwise, a message is dispatched as it is found during the directory search.

MMBXNAME is the name of the mailbox. If this is null, then the user's login name is used. See also MDLVRDIR and MMBXPROT.

MMBXPREF is a string written before the message is written into the mailbox. It is usually set to a sequence of CTRL-A characters. See also MMBXSUFF.

MMBXPROT gives the protection mode in octal for the user's mailbox. See also MDLVRDIR and MMBXNAME.

MMBXSUFF is a string written after the message is written into the mailbox. It is usually set to a sequence of CTRL-A characters. See also MMBXPREF.

MMSGLOG controls the logging information produced by *deliver* and *submit*. See also MCHANLOG, AUTHLOG, and MLOGDIR.

MMSGQ is the directory for the files of message text. If it is not a full pathname, it is taken relative to MQUEDIR.

MPHSDIR is the directory in which the timestamps for the channels are made, showing what phase of activity they are in.

MQUEDIR is the parent directory for the various queues and address directories.

MQUEPROT gives the protection mode in octal that the parent of the MQUEDIR directory should have.

MSIG is the signature that MMDF uses when notifying senders of mail delivery problems.

MSLEEP is the length of time in seconds that the background delivery daemon sleeps between queue scans.

MSUPPORT is the address to which to send mail that MMDF cannot cope with (i.e., that MMDF cannot deliver or return to its sender).

MTBL defines an alias, domain, or channel table. The following parameters can be used:

name	a short name by which the table can be referred to later in the file
file	the file from which the contents of the table are built
show	a display line, which is used for pretty printing purposes to explain what the table is all about

A typical example might be:

```
MTBL name=aliases, file=aliases,
     show="User & list aliases"
```

If the first argument does not have an equal sign, the values of the other parameters take on this value. The following example sets the *name*, *file*, and *show* parameters to the string "aliases", then resets the *show* parameter to the string "Alias table".

```
MTBL aliases, show="Alias table"
```

MTBLDIR is the default directory for the table files.

MTEMPT is the temporary files directory. If it is not a full pathname, it is taken relative to MQUEDIR.

MWARNTIME specifies the time in hours that a message can remain in a queue before a warning message about delayed delivery is sent to the sender. See also MFAILTIME.

UUname defines the UUCP sitename (short form, not full path) and is used only by the UUCP channel. See also MLNAME.

See Also

dbmbuild(ADM), mmdf(ADM), tables(F), "Setting Up Electronic Mail" in the *System Administrator's Guide*

Value Added

mmdftailor is an extension of AT&T System V provided in Altos UNIX System V.

mnttab

format of mounted filesystem table

Syntax

#include <stdio.h>
#include <mnttab.h>

Description

The **/etc/mnttab** file contains a table of devices mounted by the *mount*(ADM) command.

Each table entry contains the pathname of the directory on which the device is mounted, the name of the device special file, the read/write permissions of the special file, and the date on which the device was mounted.

The maximum number of entries in *mnttab* is based on the system parameter NMOUNT, which defines the number of allowable mounted special files.

See Also

mount(ADM)

mtune

tunable parameter file

Syntax

/etc/conf/cf.d/mtune

Description

The **mtune** file contains information about all the system tunable parameters. Each tunable parameter is specified by a single line in the file, and each line contains the following whitespace-separated set of fields:

1. *parameter name*: A character string no more than 20 characters long. It is used to construct the preprocessor "#define's" that pass the value to the system when it is built.

2. *default value*: This is the default value of the tunable parameter. If the value is not specified in the **stune** file, this value will be used when the system is built.

3. *minimum value*: This is the minimum allowable value for the tunable parameter. If the parameter is set in the **stune** file, the configuration tools will verify that the new value is equal to or greater than this value.

4. *maximum value*: This is the maximum allowable value for the tunable parameter. If the parameter is set in the **stune** file, the configuration tools will check that the new value is equal to or less than this value.

The file **mtune** normally resides in **/etc/conf/cf.d**. However, a user or an add-on package should never directly edit the **mtune** file to change the setting of a system tunable parameter. Instead the *idtune* command should be used to modify or append the tunable parameter to the **stune** file.

In order for the new values to become effective, the UNIX system kernel must be rebuilt and the system must then be rebooted.

See Also

stune(F), idbuild(ADM), idtune(ADM)

mvdevice

video driver backend configuration file

Syntax

/etc/conf/cf.d/mvdevice

Description

The *mvdevice* file accomplishes configurability of video hardware by permitting the linking of back ends to the console video driver. This linking scheme includes a C library of video back ends for use with the link kit and separate driver entries for each of the back ends.

The configuration program uses the *mvdevice* file to produce a **space.c** for the console driver. This **space.c** includes the appropriate include files and extern references to the appropriate video back ends. In addition, the configuration program builds the console display switch with in the **space.c.**

The *mvdevice* file has the following entries:

prefix	name of driver from 1 to 4 characters long (for example "mono") This is the name of the video back end.
routine_mask	This mask tells which routines were supported by the particular back end. These routines are: vvvinit(), vvvcmos(), vvvinitscreen(), vvvscroll(), vvvcopy(), vvvclear(), vvvpchar(), vvvscurs(), vvvsgr(), vvvioctl(), vvvadapctl().
type	This is placed in the file as a literal. for example, if the word MONO was put into the file, it would include the word MONO as the type entry of the adapter structure.
oem	OEM information treated exactly the same as the type (i.e. a literal)
paddr	the physical address which the video RAM is located. This would allow a user to configure a future driver. Also included as a literal field.

size — The size of the video RAM. Also included as a literal field.

This information provides all the basic information needed for the program to generate an appropriate **space.c** and build the the correct adapter switch.

The routine mask uses the following bits to signify the following routines:

0x0001	vvvinit()
0x0002	vvvcmos()
0x0004	vvvinitscreen()
0x0008	vvvscroll()
0x0010	vvvcopy()
0x0020	vvvclear()
0x0040	vvvpchar()
0x0080	vvvscurs()
0x0100	vvvsgr()
0x0200	vvvioctl()
0x0400	vvvadapctl()

The default mvdevice file looks like this:

```
#
#       mvdevice:       video configuration master file.
#
#
#prefix name    routinestype    oem     paddr   size
mono    MONO    0x07fd          MONO    0       0       0
cga     CGA     0x07fd          CGA     0       0       0
ega     EGA     0x07ff          EGA     0       0       0
vga     VGA     0x07ff          VGA     0       0       0
```

See Also

sdevice(F)

nl_types

data types for native language support

Syntax

#include <nl_types.h>

Description

This is a header file that provides type definitions used in the native language support interface. The types are:

nl_item
A type defintion for an item of language data as used by *nl_langinfo*(S). The values for **nl_item** are defined in the file **<langinfo.h>**.

nl_catd
A message catalogue descriptor. (Message catalogues are not currently supported.)

See Also

langinfo(F), nl_langinfo(S)

Value Added

nl_types is an extension of AT&T System V provided in Altos UNIX System V.

null

the null file

Description

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

Files

/dev/null

Standards Conformance

null is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

passwd

the password file

Description

The **passwd** file contains the following information for each user:

-Login name

-Numerical user ID

-Numerical group ID

-Comment

-Initial working directory

-Program to use as shell

Refer to *finger*(C) for information in the required format of the comment field for *finger*(C) to display the information. Each user is separated from the next by a newline. If the password field is null, no password is demanded; if the shell field is null, *sh*(C) is used.

This file resides in the directory **/etc**. Encrypted passwords are not stored in **/etc/passwd**.

Warning

Under no circumstances should you edit **/etc/passwd** with a text editor. This will cause a series of error messages to be displayed and could prevent any further logins. Use the *sysadmsh* Accounts selection to modify or add user accounts.

Files

/etc/passwd

See Also

login(M), passwd(C), a64l(S), getpwent(S), group(F)

Standards Conformance

acct is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
The X/Open Portability Guide II of January 1987;
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

permissions

format of UUCP Permissions file

Description

The **Permissions** file (**/usr/lib/uucp/Permissions**) specifies the permissions for remote computers concerning login, file access, and command execution. In the **Permissions** file, you can specify the commands that a remote computer can execute and restrict its ability to request or receive files queued by the local site.

Each entry is a logical line with physical lines terminated by a \ to indicate continuation. Entries are made up of options delimited by white space. Each option is a name-value pair in the following format:

name=value

Note that no white space is allowed within an option assignment.

Comment lines begin with a pound sign (#) and they occupy the entire line up to a newline character. Blank lines are ignored (even within multi-line entries).

There are two types of **Permissions** file entries:

LOGNAME	specifies the permissions that take effect when a remote computer calls your computer.
MACHINE	specifies permissions that take effect when your computer calls a remote computer.

Examples

This entry is for public login. It provides the default permissions. Note that use of this type of anonymous login is not encouraged.

```
LOGNAME=nuucp \
MACHINE=OTHER \
READ=/usr/spool/uucppublic \
WRITE=/usr/spool/uucppublic \
SENDFILES=call REQUEST=no \
COMMANDS=/bin/rmail
```

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C)

plot

graphics interface

Description

Files of this format are produced by routines described in *plot*(S) and are interpreted for various devices by commands described in *tplot*(ADM). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the **x** and **y** values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *plot*(S).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes [see *tplot*(ADM)].

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.

e erase: Start another frame of output.

f linemod: Take the following string, up to a new-line, as the style for drawing further lines. The styles are "dotted", "solid", "longdashed", "shortdashed", and "dotdashed". Effective only for the **-T4014** and **-Tver** options of *tplot*(ADM) (TEKTRONIX 4014 terminal and VERSATEC plotter).

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *tplot*(ADM). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
TEKTRONIX 4014	space(0, 0, 3120, 3120);
VERSATEC plotter	space(0, 0, 2048, 2048);

See Also

plot(S), term(M), graph(ADM), tplot(ADM)

Notes

The plotting library *plot*(S) and the curses library *curses*(S) both use the names erase() and move(). The curses versions are macros. If you need both libraries, put the *plot*(S) code in a different source file than the *curses*(S) code, and/or #undef move() and erase() in the *plot*(S) code.

pnch

file format for card images

Description

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

poll

format of UUCP Poll file

Description

The **Poll** file (**/usr/lib/uucp/Poll**) contains information for polling remote computers. Each entry in the **Poll** file contains the name of a remote computer to call, followed by a tab character, and the hours the computer should be called. The hours must be integers in the range 0-23.

Poll file entries have the following format:

sysname<TAB>*hour* ...

The following entry provides polling of computer *gorgon* every four hours:

```
gorgon  0 4 8 12 16 20
```

The **uudemon.poll** script controls polling but does not perform the poll. It sets up a polling file (**C.sysnxxxx**) in the /usr/spool/uucp/*nodename* directory, where *nodename* is replaced by the name of the machine. This file will in turn be acted upon by the scheduler (started by **uudemon.hour**). The **uudemon.poll** script is scheduled to run semi-hourly before **uudemon.hour** so that the work files will be there when **uudemon.hour** is called. The default root **crontab** entry for **uudemon.poll** is as follows:

```
1,30 * * * * "/usr/lib/uucp/uudemon.poll > /dev/null"
```

See Also

uucico(ADM), uucp(C), cron(C), crontab(C)

Standards Conformance

poll is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

purge

the policy file of the sanitization utility purge(C)

Syntax

/etc/default/purge

Description

This file is an ASCII file whose lines each designate a file, filesystem or device to be a member of a *type*. The command:

purge -t *type*

would overwrite all the members of *type*.

Blank lines and lines beginning with '#' are ignored. Entries are of the form:

file *type* [*count*]

This specifies that *file* is a member of *type*. The optional field *count* is the number of times to overwrite *file* when it is purged. The default is one.

The two *types* **system** and **user** are hardwired into the *purge*(C) utility. These types can be overwritten with the **-s** and **-u** switches to *purge* respectively.

This file should be configured on site to reflect files and devices which are sensitive and need to be protected from unauthorized access.

The initial contents of the file is:

```
/tmp     system
/user    user
```

Files

/etc/default/purge

See Also

purge(C), sysadmsh(ADM)

Value Added

purge is an extension of AT&T System V provided in Altos UNIX System V.

queue

MMDF queue files for storing mail in transit

Syntax

/usr/spool/mmdf/lock/home

Description

MMDF stores mail in an isolated part of the file system, so that only authorized software may access the mail. Mail is stored under the directory sub-tree.

It must specify a path with at least two sub-directories. The next-to-bottom one is used as a "locking" directory and the bottom one is the **home**. For full protection, only authorized software can move through the locking directory. Hence, it is owned by MMDF and accessible only to it.

Queue Entries

When mail is queued by *submit*, it is actually stored as two files. One contains the actual message text and the other contains some control information and the list of addressees.

The text file is stored in the **msg** directory. The other file is stored in the **addr** directory and is linked into one or more queue directories. The queue directories are selected based on the channels on which this message will be delivered. Each output channel typically has its own queue directory.

Another directory below **home** is a temporary area called **tmp**. It holds temporary address-lists as they are being built. Queuing of a message is completed by linking this address file into **addr** and the queue directories. The **msg** directory contains files with message text. **Addr** and **msg** files are synchronized by giving them the same unique name, which MMDF occasionally calls the message "name". The message name is derived by use of *mktemp(S)*, using **msg** as the base directory. The files created in **addr** must be open read-write access; the ones in **msg** must be open read access.

When *submit* runs, it changes into **home** for its working directory. It then does a *setuid*() to run as the caller. This is necessary to permit *submit* to access the caller's address-list files (specified as "< file"), but probably will be changed. *Deliver* changes into the queue directory to minimize the time spent searching for messages to deliver.

The following depicts the directory organization:

```
                         lock           (lock: 0700)
                           |            (mmdf only)
                           |
               _________home______      (open: 0777)
              /     |        |      \
             /      |        |       \
          tmp     addr      q.*      msg    (open: 0777)

     addresses ==> moved   and linked  message text
     built here ==> into here  into here  put here

     entries:      0666       0666       0644
```

Queue File Formats

The **msg** portion of a queued message simply contains the message, which must conform to the Arpanet standard syntax, specified in RFC822. It is expected that the format of the message contents file eventually will be more structured, permitting storage of multi-media mail.

The following specifies the syntax of the **addr** (and queue directory) address-list portion of the queued message:

Address File Contents

file ::=	creation late flags '\n' [rtrn-addr] '\n' *(adr_queue '\n')
creation ::=	{long integer decimal representation of time message was created}
late ::=	ADR_MAIL / ADR_DONE {from adr_queue.h}
flags ::=	{decimal representation of 16-bits of flags}
rtrn-addr ::=	{rfc822 return address}
adr_queue ::=	temp_ok mode queue host local {conforms to structure specified in adr_queue.h}
temp_ok ::=	{temporary flag indicating whether this address has been verified by the receiving host: "yes" is "+"; "not yet" is "-"}

mode ::=	{send to mailbox(m), tty(t), both(b), either(e), or processing completed(*)}
queue ::=	{name of the queue into which this message is linked for this address}
host ::=	{official name (and domain) of recipient host}
local ::=	{local address on receiving host}

Address File Description

An address queue file contains a creation time-stamp, an indication if the sender has been notified of delayed delivery, some flags, an optional return-mail address, and an address list. Several <flags> are currently in use (as specified in msg.h). ADR_NOWARN indicates whether late warnings should be sent to the return-mail address if the entire address list has not been processed within the number of hours specified by "warntime". ADR_NORET indicates whether mail should be returned to the sender if it hasn't been completely processed within the number of hours specified by "failtime". ADR_CITE indicates whether warning and failure messages are to contain only a citation of the message. An ADR_DONE ("*") value, for the "late" flag, indicates that a warning notice has been sent.

The creation date is coded as a long ASCII decimal string, terminated by the "late" flag. This has to be stuffed into the file because Unix doesn't maintain it. The date is used to sort the queue so that mail is delivered in the order submitted.

The return address is a line of text and may be any address acceptable to *submit*.

Each address entry is on a separate line, and conforms to the adr_struct format, defined in *adr_queue.h*. It contains several fields separated by spaces or commas. Fields containing spaces or commas must be enclosed in double quotes.

The temp_ok flag indicates whether the address has been accepted during an "address verification" dialog with the receiving host. When the message text is successfully accepted by the receiving host, then this flag no longer applies and the mode is set to ADR_DONE ("*"). Before final acceptance of message text, the mode flag indicates whether the mail is for a mailbox, terminal, both, or either. (Currently only mailbox delivery, ADR_MAIL, is used.)

The queue name is the name of the sub-queue in which the message is queued for this address. Each addressee's host may be on a separate queue or some hosts may share the same queue. When all addressees in the same queue have been delivered, the address file is removed from that queue directory. When all queues have been processed, the address file (in both **addr** and the queue directory) and the text file (in **msg**) are removed.

The host and local strings are used by the channel program. The host determines the type of connection the channel program makes. The local string is passed to the host; it should be something meaningful to that host. The local string must not contain newline or null and it be a valid local address per RFC822.

See Also

deliver(ADM), cleanque(ADM), submit(ADM)

queuedefs

scheduling information for cron queues

Description

The **queuedefs** file is read by the clock daemon, *cron*, and controls how jobs submitted with *at*, *batch*, and *crontab* are executed. Every job submitted by one of these programs is placed in a certain queue, and the behavior of these queues is defined in **/usr/lib/cron/queuedefs**. Queues are designated by a single, lowercase letter. The following queues have special significance:

a	*at* queue
b	*batch* queue
c	*cron* queue

For a given queue, the **queuedefs** file specifies the maximum number of jobs that may be executing at one time (*njobs*), the priority at which jobs will execute (*nice*), and the how long *cron* will wait between attempts to run a job (*wait*). If *njobs* jobs are already running in a given queue when a new job is scheduled to begin execution, *cron* will reschedule the job to execute *wait* seconds later. A typical file might look like this:

```
a.4j1n
b.2j2n90w
```

Each line gives parameters for one queue. The line must begin with a letter designating a queue, followed by a period (.). This is followed by the numeric values for *njobs*, *nice*, and *wait*, followed respectively by the letters "j", "n", and "w". The values must appear in this order, although a value and its corresponding letter may be omitted entirely, in which case a default value is used. The default values are *njobs* = 100, *nice* = 2, and *wait* = 60.

The value for *nice* is added to the default priority of the job (a higher numerical priority results in a lower scheduling priority - see *nice*(C)). *wait* is given in seconds.

Files

/usr/lib/cron/queuedefs	**queuedefs** file

reloc

relocation information for a common object file

Syntax

#include <reloc.h>

Description

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format.

```
struct   reloc
{
         long     r_vaddr ;    /* (virtual) address of
                                  reference */
         long     r_symndx ;   /* index into symbol table */
         short    r_type ;     /* relocation type */
} ;
# define R_PCRLONG  024
```

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated.

R_PCRLONG A "PC-relative" 32-bit reference to the symbol's virtual address.

More relocation types exist for other processors. Equivalent relocation types on different processors have equal values and meanings. New relocation types will be defined (with new values) as they are needed.

Relocation entries are generated automatically by the assembler and automatically used by the link editor. Link editor options exist for both preserving and removing the relocation entries from object files.

See Also

as(CP), ld(CP), a.out(F), syms(F)

sccsfile

format of an SCCS file

Description

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines). Each logical part of an SCCS file is described in detail below.

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character. Entries of the form DDDDD represent a five digit string (a number between 00000 and 99999).

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The @hR provides a *magic number* of (octal) 064001.

Delta Table

The delta table consists of a variable number of entries of the form:

@s DDDDD/DDDDD/DDDDD
@d <type> **<SCCS ID>** yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
@c <comments> ...
.
.
@e

The first line (@s) contains the number of lines inserted/deleted/unchanged respectively. The second line (@d) contains the type of the delta (currently, normal: D, and removed: R), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The @i, @x, and @g lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The @m lines (optional) each contain one MR number associated with the delta; the @c lines contain comments associated with the delta.

The @e line ends the delta table entry.

User Names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

Flags

Keywords used internally (see *admin*(CP) for more information on their use). Each flag line takes the form:

@f <flag> <optional text>

The following flags are defined:

@f t	<type of program>
@f v	<program name>
@f i	
@f b	
@f m	<module name>
@f f	<floor>
@f c	<ceiling>
@f d	<default-sid>
@f n	
@f j	
@f l	<lock-releases>
@f q	<user defined>

The **t** flag defines the replacement for the identification keyword. The v flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the **b** flag is present the **-b** option may be used with the *get* command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the sccsfile.F identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a *get* command. The **n** flag causes *delta* to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes *get* to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing (*get*(CP) with the **-e** option). The **q** flag defines the replacement for the identification keyword.

Comments

Arbitrary text surrounded by the bracketing lines @t and @T. The comments section typically contains a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, as follows:

@I DDDDD
@D DDDDD
@E DDDDD

The digit string (DDDDD) is the serial number corresponding to the delta for the control line.

See Also

admin(CP), delta(CP), get(CP), prs(CP)

Standards Conformance

sccsfile is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

scnhdr

section header for a common object file

Syntax

#include <scnhdr.h>

Description

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct  scnhdr
{
        char            s_name[SYMNMLEN]; /* section name */
        long            s_paddr;      /* physical address */
        long            s_vaddr;      /* virtual address */
        long            s_size;       /* section size */
        long            s_scnptr;     /* file ptr to raw data */
        long            s_relptr;     /* file ptr to relocation */
        long            s_lnnoptr;    /* file ptr to line numbers */
        unsigned short  s_nreloc;     /* # reloc entries */
        unsigned short  s_nlnno;      /* # line number entries */
        long            s_flags;      /* flags */
} ;
```

File pointers are byte offsets into the file; they can be used as the offset in a call to FSEEK [see *ldfcn*(F)]. If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. It has a size, symbols defined in it, and symbols that refer to it. But it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for *s_scnptr*, *s_relptr*, *s_lnnoptr*, *s_nreloc*, and *s_nlnno* are zero.

See Also

ld(CP), fseek(S), a.out(F)

scr_dump

format of curses screen image file

Syntax

scr_dump(file)

Description

The *curses*(S) function *scr_dump*() will copy the contents of the screen into a file. The format of the screen image is as described below.

The name of the tty is 20 characters long and the modification time (the *mtime* of the tty that this is an image of) is of the type *time_t*. All other numbers and characters are stored as *chtype* (see **<curses.h>**). No new-lines are stored between fields.

<magic number: octal 0433>
<name of tty>
<mod time of tty>
<columns> <lines>
<line length> <chars in line>for each line on the screen
<line length> <chars in line>
.
.
.
<labels?> 1, if soft screen labels are present
<cursor row> <cursor column>

Only as many characters as are in a line will be listed. For example, if the *<line length>* is **0**, there will be no characters following *<line length>*. If *<labels?>* is TRUE, following it will be

<number of labels>
<label width>
<chars in label 1>
<chars in label 2>
.
.
.

See Also

curses(S)

sdevice

local device configuration file

Syntax

/etc/conf/cf.d/sdevice

Description

The *sdevice* file contains local system configuration information for each of the devices specified in the *mdevice* file. It contains one or more entries for each device specified in *mdevice. sdevice* is present in the directory */etc/conf/cf.d,* and is coalesced from component files in the directory */etc/conf/sdevice.d.* Files in */etc/conf/sdevice.d* are the *System* file components either delivered with the base system or installed later via *idinstall.*

Each entry must contain the following whitespace-separated fields:

1. *Device name:* This field contains the internal name of the driver. This must match one of the names in the first field of an *mdevice* file entry.

2. *Configure:* This field must contain the character 'Y' indicating that the device is to be installed in the kernel. For testing purposes, an 'N' may be entered indicating that the device will not be installed.

3. *Unit:* This field can be encoded with a device dependent numeric value. It is usually used to represent the number of subdevices on a controller or pseudo-device. Its value must be within the minimum and maximum values specified in fields 7 and 8 of the *mdevice* entry.

4. *Ipl*: The *ipl* field specifies the system ipl level at which the driver's interrupt handler will run in the new system kernel. Legal values are 0 through 8. If the driver doesn't have an interrupt handling routine, put a 0 in this field.

5. *Type:* This field indicates the type of interrupt scheme required by the device. The permissible values are:

   0 - The device does not require an interrupt line.

   1 - The device requires an interrupt line.
   If the driver supports more than one hardware controller, each controller requires a separate interrupt.

2 - The device requires an interrupt line.
If the driver supports more than one hardware controller, each controller will share the same interrupt.

3 - The device requires an interrupt line.
If the driver supports more than one hardware controller, each controller will share the same interrupt. Multiple device drivers having the same ipl level can share this interrupt.

6. *Vector:* This field contains the interrupt vector number used by the device. If the *Type* field contains a 0 (i.e., no interrupt required), this field should be encoded with a 0. Note that more than one device can share an interrupt number.

7. *SIOA:* The *SIOA* field (Start I/O Address) contains the starting address on the I/O bus through which the device communicates. This field must be within 0x1 and 0x3fff. (If this field is not used, it should be encoded with the value zero.)

8. *EIOA:* The field (End I/O Address) contains the end address on the I/O bus through which the device communicates. This field must be within 0x1 and 0x3fff. (If this field is not used, it should be encoded with the value zero.)

9. *SCMA:* The *SCMA* field (Start Controller Memory Address) is used by controllers that have internal memory. It specifies the starting address of this memory. This field must be within 0xa0000 and 0xfbfff. (If this field is not used, it should be encoded with the value zero.)

10. *ECMA:* The *ECMA* (End Controller Memory Address) specifies the end of the internal memory for the device. This field must be within 0xa0000 and 0xfbfff. (If this field is not used, it should be encoded with the value zero.)

See Also

mdevice(F), idinstall(ADM)

sfsys

local filesystem type file

Syntax

/etc/conf/cf.d/sfsys

Description

The **sfsys** file contains local system information about each file system type specified in the **mfsys** file. It is present in the directory */etc/conf/cf.d*, and contains a one-line entry for each file system type specified in the **mfsys** file. The **sfsys** file is coalesced from component files in the directory **/etc/conf/sfsys.d**. Each line in this file is a whitespace-separated set of fields that specifies:

name This field contains the internal name of the file system type (e.g., DUFST, S51K). By convention, this name is up to 32 characters long, and is composed of all uppercase alphanumeric characters.

Y/N This field contains either an uppercase "Y" (for "yes") or an uppercase "N" (for "no") to indicate whether the named file system type is to be configured into the next system kernel to be built.

See Also

mfsys(F), idinstall(ADM), idbuild(ADM)

stat

data returned by stat system call

Syntax

#include <sys/stat.h>

Description

The **sys/stat.h** include file contains the definition for the structure returned by the *stat* and *fstat* functions. The structure is defined as:

```
struct stat{
   dev_t      st_dev;     /*


   ino_t      st_ino;     /* inode number */
   ushort     sh_mode;    /* file mode */
   short      st_nlink;   /* # of links */
   ushort     st_uid;     /* owner uid */
   ushort     st_gid;     /* owner gid */
   dev_t      st_rdev;    /*


   off_t      st_size;    /* file size in bytes */
   time_t     st_atime;   /* time of last access */
   time_t     st_mtime;   /* time of last data modification */
   time_t     st_ctime;   /* time of last file status 'change' */
};
```

Note that the *st_atime*, *st_mtime*, and *st_ctime* values are measured in seconds since 00:00:00 (GMT) on January 1, 1970.

The *st_mode* value is actually a combination of one or more of the following file mode values:

```
S_IFMT     0170000    /* type of file */
S_IFDIR    0040000    /* directory */
S_IFCHR    0020000    /* character special */
S_IFBLK    0060000    /* block special */
S_IFREG    0100000    /* regular */
S_IFIFO    0010000    /* fifo */
S_IFNAM    0050000    /* name special entry */
S_INSEM    01         /* semaphore */
S_INSHD    02         /* shared memory */
S_ISUID    04000      /* set user id on execution */
S_ISGID    02000      /* set group id on execution */
S_ISVTX    01000      /* save swapped text even after use */
S_IREAD    00400      /* read permission, owner */
S_IWRITE   00200      /* write permission, owner */
S_IEXEC    00100      /* execute/search permission, owner */
```

Files

/usr/include/sys/stat.h

See Also

stat(S)

Standards Conformance

stat is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
The X/Open Portability Guide II of January 1987;
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

stune

local tunable parameter file

Syntax

/etc/conf/cf.d/stune

Description

The **stune** file contains local system settings for tunable parameters. The parameter settings in this file replace the default values specified in the **mtune** file, if the new values are within the legal range for the parameter specified in **mtune**. The file contains one line for each parameter to be reset. Each line contains two whitespace-separated fields:

1. *external name*: This is the external name of the tunable parameter used in the **mtune** file.

2. *value*: This field contains the new value for the tunable parameter.

The file **stune** normally resides in **/etc/conf/cf.d.** However, a user or an add-on package should never directly edit the **mtune** file. Instead the *idtune* command should be used.

In order for the new values to become effective the Altos UNIX System V kernel must be rebuilt and the system must then be rebooted.

See Also

mtune(F), idbuild(ADM), idtune(ADM)

syms

common object file symbol table format

Syntax

#include <syms.h>

Description

Common object files contain information to support symbolic software testing [see *sdb*(CP)]. Line number entries, *linenum*(F), and extensive symbolic information permit testing at the C *source* level. Every object file's symbol table is organized as shown below.

```
File name 1.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 1.

File name 2.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 2.
...

Defined global symbols.
Undefined global symbols.
```

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure is given below.

```
#define  SYMNMLEN  8
#define  FILNMLEN  14
#define  DIMNUM    4

struct  syment
{
    union                           /* all ways to get symbol name */
    {
        char       _n_name[SYMNMLEN]; /* symbol name */
        struct
```

```
        {
            long        _n_zeroes;     /* == 0L when in string table */
            long        _n_offset;     /* location of name in table */
        } _n_n;
        char            *_n_nptr[2];   /* allows overlaying */
    } _n;
    long                n_value;       /* value of symbol */
    short               n_scnum;       /* section number */
    unsigned short      n_type;        /* type and derived type */
    char                n_sclass;      /* storage class */
    char                n_numaux;      /* number of aux entries */
};

#define  n_name    _n._n_name
#define  n_zeroes  _n._n_n._n_zeroes
#define  n_offset  _n._n_n._n_offset
#define  n_nptr    _n._n_nptr[1]
```

Meaningful values and explanations for them are given in both **syms.h** and *Common Object File Format*. Anyone who needs to interpret the entries should seek more information in these sources. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent
{
        struct
        {
                long            x_tagndx;
                union
                {
                        struct
                        {
                                unsigned short  x_lnno;
                                unsigned short  x_size;
                        } x_lnsz;
                        long    x_fsize;
                } x_misc;
                union
                {
                        struct
                        {
                                long    x_lnnoptr;
                                long    x_endndx;
                        }       x_fcn;
                        struct
                        {
                                unsigned short  x_dimen[DIMNUM];
                        }       x_ary;
                }               x_fcnary;
                unsigned short  x_tvndx;
        }       x_sym;
        struct
        {
```

```
                char   x_fname[FILNMLEN];
        }       x_file;
         struct
         {
                  long    x_scnlen;
                  unsigned short  x_nreloc;
                  unsigned short  x_nlinno;
         }        x_scn;

        struct
        {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        }       x_tv;
};
```

Indexes of symbol table entries begin at *zero*.

See Also

sdb(CP), a.out(F), linenum(F).

Notes

On machines on which **ints** are equivalent to **longs**, all **longs** have their type changed to **int.** Thus the information about which symbols are declared as **longs** and which, as **ints**, does not show up in the symbol table.

sysfiles

format of UUCP Sysfiles file

Description

The **/usr/lib/uucp/Sysfiles** file lets you assign different files to be used by *uucp*(C) and *cu*(C) as **Systems, Devices,** and **Dialers** files.

You can use different **Systems** files so that requests for login services can be made to different addresses than UUCP services.

With different **Dialers** files you can use different handshaking for *cu* and *uucp*. Multiple **Systems, Dialers,** and **Devices** files are useful if any one file becomes too large.

An active **Sysfiles** file is not included in the distribution. Instead a **Sysfiles.eg** file is included, which contains comments and commented examples of how such a file can be used. This is done because UUCP runs faster without reading this file.

The format of the **Sysfiles** file is

service=w systems=x:x dialers=y:y devices=z:z

where *w* is replaced by *uucico*(ADM), *cu*, or both separated by a colon; *x* is one or more files to be used as the **Systems** file, with each file name separated by a colon and read in the order presented; *y* is one or more files to be used as the **Dialers** file; and *z* is one or more files to be used as the **Devices** file. Each file is assumed to be relative to the **/usr/lib/uucp** directory, unless a full path is given. A backslash-carriage return (\<*CR*>) can be used to continue an entry on to the next line.

An example of using a local *Systems* file in addition to the usual *Systems* file follows:

```
service=uucico:cu  systems=Systems:Local_Systems
```

If this is in */usr/lib/uucp/Sysfiles*, then both **uucico** and **cu** will first look in */usr/lib/uucp/Systems*. If the system they're trying to call doesn't have an entry in that file, or if the entries in the file fail, then they'll look in */usr/lib/uucp/Local_Systems*.

When different *Systems* files are defined for **uucico** and **cu** services, your machine will store two different lists of Systems. You can print the **uucico** list using the *uuname* command or the *cu* list using the *uuname -c* command.

Examples

The following example uses different **Systems** and **Dialers** files to separate the *uucico* and *cu*-specific info, with information that they use in common still in the ''usual'' **Systems** and **Dialers** files.

```
service=uucico  systems=Systems.cico:Systems \
                dialers=Dialers.cico:Dialers
service=cu      systems=Systems.cu:Systems \
                dialers=Dialers.cu:Dialers
```

This next example uses the same systems files for uucico and cu, but has split the **Systems** file into local, company-wide, and global files.

```
service=uucico  systems=Systems.local:Systems.company:Systems
service=cu      systems=Systems.local:Systems.company:Systems
```

See Also

uucico(ADM), uucp(C), systems(F)

systemid

the Micnet system identification file

Description

The **systemid** file contains the machine and site names for a system in a Micnet network. A *machine name* identifies a system and distinguishes it from other systems in the same network. A *site name* identifies the network to which a system belongs and distinguishes the network from other networks in the same chain.

The **systemid** file may contain a *site name* and up to four different *machine names*. The file has the form:

[site-name]
[machine-name1]
[machine-name2]
[machine-name3]
[machine-name4]

The file must contain at least one machine name. The other machine names are optional, serving as alternate names for the same machine. The file must contain a site name if more than one machine name is given or if the network is connected to another through a uucp link. The site name, when given, must be on the first line.

Each name can have up to eight letters and numbers but must always begin with a letter. There is never more than one name to a line. A line beginning with a pound sign (#) is considered a comment line and is ignored.

The Micnet network requires one **systemid** file on each system in a network with each file containing a unique set of machine names. If the network is connected to another network through a uucp link, each file in the network must contain the same site name.

The **systemid** file is used primarily during resolution of aliases. When aliases contain site and/or machine names, the name is compared with the names in the file and removed if there is a match. If there is no match, the alias (and associated message, file, or command) is passed on to the specified site or machine for further processing.

Files

/etc/systemid

See Also

aliases(M), netutil(ADM), top(F)

Value Added

systemid is an extension of AT&T System V provided in Altos UNIX System V.

systems

format of UUCP Systems file

Description

The **Systems** file (**/usr/lib/uucp/Systems**) contains the information needed by the *uucico* daemon to establish a communication link to a remote computer. Each entry in the file represents a computer that your computer can call. You can configure the **Systems** file to prevent unauthorized computers from logging in on your computer. More than one entry may be present for a particular computer. These additional entries represent alternative communication paths which the computer tries in sequential order.

Each entry in the *Systems* file has the following format:

sitename schedule device speed phone login-script

sitename field contains the node name of the remote computer.

schedule field is a string that indicates the day-of-week and time-of-day when the remote computer can be called.

device is the device type that should be used to establish the communication link to the remote computer.

speed indicates the transfer speed of the device used in establishing the communication link.

phone provides the phone number of the remote computer for automatic dialers. If you wish to create a portable *Systems* file that can be used at a number of sites where the dialing prefixes differ, see the *dialcodes*(F) man page.

login-script contains login information (also known as a ''chat script'').

See Also

uucico(ADM), uucp(C), devices(F), dialers(F)

tables

MMDF name tables for aliases, domains, and hosts

Description

All of the MMDF name tables are encoded into a database which is built on top of the *dbm*(S) package. A number of tables are associated with MMDF, the exact set being specified by the tailor file, */usr/mmdf/mmdftailor*. Name tables all have the same format. Functionally, they permit a simple key/value pairing. The syntax for tables is specified here:

entries ::=	entries entry
entry ::=	comment / real-entry
comment ::=	'#' value eol
real-entry ::=	name separator value eol
name ::=	{string of chars not containing a <separator>}
separator ::=	{see the chars in _hkeyend[], usually ':' and space}
value ::=	{string of chars not containing an <eol>}
eol ::=	{see the chars in _hvalend[]}
where:	
name is	a key
value is	any relevant text.

Hosts and Domains

Two basic types of table are host and domain tables. This section gives a brief discussion of these concepts in terms of the MMDF system. The domain namespace is treated as a logical global hierarchy, according to the model of RFC 819, with subdomains separated by '.'s (e.g ISI.USC.ARPA is a three level hierarchy with ARPA at the top level). A host is a computer associated with a channel which may be directly connected or reached through a relay associated with the channel. The distinction between hosts as physical entities, and domains as logical entities should be noted. All hosts known to an MMDF system must have unique names. For this reason, the convention of labelling hosts by an associated domain name is adopted in many cases. This is a useful method to guarantee unique names, but is not required. The domain and host table structures are devised with three basic aims in mind:

1. To map a string into a fully expanded domain name.

2. To map this domain into a source route starting with a host.

3. To obtain the transport address associated with the host.

Domain Tables

Domains are split in a two-level manner, with the top part of the tree specified in the tailor file and the lower parts of the tree in tables. The two level structure is intended as a balance between generality and efficiency. The order of searching is also specified in the tailor file. The structure of a domain table is to have *name* as the part of the domain not in the tailor file. Thus for ISI.USC.ARPA there might be a domain ARPA with name=isi.usc or domain USC.ARPA with name=isi. The structure of value is:

value ::=	*(domain dm_separator) host

The possible values of dm_separator are given in tai(S), although typically ',' or ' ' would be used. This value is essentially a source route to be traversed from right to left. Consider an example table for the domain ARPA:

```
# Sample ARPA domain table
isi.usc:a.isi.usc.arpa
b.isi.usc:b.isi.usc.arpa
foobar.isi.usc:b.isi.usc.arpa
graphics.isi.usc:graphics.isi.usc.arpa z.mit.arpa
```

Thus, if the "isi.usc.arpa" is analyzed, domain table ARPA will be selected, and host "a.isi.usc.arpa" associated with domain "isi.usc.arpa." If only "isi.usc" were given, the domain tables would be searched in order, and if the ARPA table were the first one to give a match, then the same result would be reached. If "foobar.isi.usc" is given, it would be mapped to host "b.isi.usc.arpa" and (official) domain "b.isi.usc.arpa." If "graphics.isi.usc.arpa" is given, a source route to domain "graphics.isi.usc.arpa" through HOST "z.mit.arpa" will be identified. If "xy.isi.usc.arpa" (or "xy.isi.usc") is given, then it will not be found. However, a subdomain will be stripped from the left and the search repeated. Thus domain "xy.isi.usc.arpa" will be identified as reached by a source route through host "a.isi.usc.arpa."

As specified earlier, the order of searching is also specified in the tailor file. For example, a host in domain UCL-CS.AC.UK, might have a search order UCL-CS.AC.UK, AC.UK, UK, SWEDEN, ARPA,

"". Thus, if there were a domain isi.usc.ac.uk, it would be the preferred mapping for isi.usc over isi.usc.arpa. The last domain searched is null. This could be used to contain random fully qualified domains or to identify gateways to other domains. An example file is:

```
#  Sample Top level domain table
#  Odd host
basservax.australia:basservax.australia scunthorpe.ac.uk
# UUCP Gateway
uucp:seismo.arpa
# Mailnet Gateway  (-> multics -> educom ->mailnet)
mailnet:educom.mailnet mit-multics.arpa
```

To specify the top domain in the tailor file, the name and dmn parameters of the domain should be set to "".

Host Tables

For every host associated with the channel, there will be one or more entries. In each case, the key is the name identified from the domain tables. A host may have multiple entries if it has more than one transport address which the channel might utilise.

When a channel just sends all its mail to a relaying site, the address portion (value) of the entry is not needed, directly, during the transmission process. Hence, it need not be accurate. However, it still is used to logically collect together host names, that is, all table entries with the same value are regarded as being aliases for the same host.

P.O. Box Channels

POBox channels, for passive, telephone-based exchange, operate in two modes. In one case, a single login is authorized to pickup all mail for the channel. In this case, the host-table addresses are only used for the "collecting" function. For the second mode, different logins share the channel and are to receive only some of the mail queued for the channel. In this case, the login is treated as an "address", and the table entries should have the value fields contain the name of the login authorized to pickup mail for that "host".

Access control tables

Channels also have access control tables associated with them, to determine whether a message is allowed to use a given route. Each channel has four (optional) tables that determine the access controls used: insrc, outsrc, indest, and outdest.

Reformatting tables

There may also be a ''known hosts'' table associated with each channel. This is exactly the same format as a host table. If a message is being reformatted, and if an address does not have its host in this list, then it will be modified to appear as a percent route (RFC733 or JNT Mail route) address, with the local domain as the root.

Local Aliases

The password file specifies the name of all local recipients; their mailing names are their login names. Since this is a rather restricted name space, and since it is useful to have some other kinds of locally-known names, there is a second file used to specify "aliases". The location of the aliases file is specified in the tailor file.

An alias entry may be used for one of five functions:

1. True aliasing, where the key value maps to a local user's login name, e.g. ''dave:dcrocker;''

2. Forwarding, where the key value maps to a foreign address, such as ''dcrocker:dcrocker@udel;'' and

3. Address lists, where the key value maps to a set of addresses, such as ''mother:cotton,dcrocker,farber.''

4. Redirection of a message to a file. For example, ''foobar:dpk/foobar'' would cause user and group ids to be set to dpk and the text of the message to be appended to the file ''foobar'' in dpk's default login directory. Similarly, ''foobar:dpk//tmp/foobar'' would do the same for file /tmp/foobar.

5. Redirection of a message to a pipe. For example, ''news-inject:news|/usr/lib/news/uurec'' would cause a message to be passed into an Altos UNIX System V pipe (see *pipe*(S)) with userid and groupid set to news.

As a convenience, the value-part of an entry may specify a file name, so that the *actual* value is taken from the file. There are two possible notations for this:

1. By having left-angle bracket ('<') precede the value specification. For example: ''mother: < /etc/mmdf/mother_list@udel-relay.arpa.''

2. By using a data type with value ''include.'' For example: ''mother: :include: /etc/mmdf/mother@udel-relay.arpa''

In both cases, the @HOST (not a domain) is optional. If specified, it should be the local host.

Recursive specification is permitted. For example, ''crocker'' may map to ''dcrocker'' and ''dcrocker'' may map to ''dcrocker at udel,'' so that both ''crocker'' and ''dcrocker'' are locally-known names, but mail sent to either of them will be forwarded to ''dcrocker@udel.''

In practice, it is useful to organize alias files into the following ordering:

List aliases
: which contain a value referring to a later address list. This constitutes a one-to-one mapping of a key to a value, where the value points into the ''Lists'' group.

Lists
: which contain values referring to multiple addresses; This constitutes a one-to-many mapping, where some of the addresses may refer to other entries (address lists) in the Lists group, as well as other entries (individual addresses) later in the table.

Mailbox aliases
: which contain values referring to single addresses. These constitute one-to-one mappings, where the value refers to an entry in the password file or to an entry in the ''Forwarding aliases'' group.

Forwarding aliases
: which contain values referring to single addresses on other machines. These, also, are one-to-one mappings, where the value always refers to an off-machine address.

By organizing the file in this manner, only the ''Lists'' portion requires a topological sort. Since the other three sections will never point to entries within their section, they may be sorted more conveniently, such as alphabetically. Such a structure also tends to make changes easy. In particular, the handling of forwarding is easy, since *all* references to a user will get intercepted, at the end of the table.

Mail-ID tables

The Mail-ID tables are used only if the Mail-IDs feature is enabled. This can be done in the tailoring file, by defining MMAILID to be 1. Mail-IDs are used to disassociate mail addresses from login names.

There are two tables that are used to map Mail-IDs to users login names and login ids to Mail-IDs. The "users" file is used to map users (login ids) to Mail-IDs, and the "mailids" file is used to map Mail-IDs to users. The names of these files can be overridden, but it is not recommended. Each file has lines with two entries per line (user and Mail-ID, or Mail-ID and user).

A user can have more than one entry in the Mail-IDs file, but should have only one entry in the users file. This does not prevent them from sending mail with any of their Mail-IDs. The users file is just a source of default Mail-IDs.

Value Added

tables is an extension of AT&T System V provided in Altos UNIX System V.

tar

archive format

Description

The command *tar*(C) dumps files to and extracts files from backup media or the hard disk.

Each file is archived in contiguous blocks, the first block being occupied by a header, whose format is given below, and the subsequent blocks of the files occupying the following blocks. All headers and file data start on 512 byte block boundaries and any spare unused space is padded with garbage. The format of a header block is as follows:

```
#define TBLOCK 512
#define NBLOCK 20
#define NAMSIZ 100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
                char extno[4];
                char extotal[4];
                char efsize[12];
        } dbuf;
} dblock;
```

The name entry is the path name of the file when archived. If the pathname starts with a zero word, the entry is empty. It is at most 100 bytes long and ends in a null byte. Mode, uid, gid, size, and time modified are the same as described under i-nodes (refer to *filesystem* (F)). The checksum entry has a value such that the sum of the words of the directory entry is zero.

If the entry corresponds to a link, then *linkname* contains the pathname of the file to which this entry is linked and *linkflag* is set to 0 if there are no links, or 1 if there are links. No data is put in the archive file.

See Also

filesystem(F), tar(C)

Standards Conformance

tar is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

term

terminal driving tables for nroff

Description

nroff uses driving tables to customize its output for various types of output devices, such as printing terminals, special word-processing printers (such as Diablo, Qume, or NEC Spinwriter mechanisms), or special output filter programs. These driving tables are written as C programs, compiled, and installed in **/usr/lib/term/tab*name***, where *name* is the name for that terminal type as shown in **term**.

The structure of the tables is as follows. Sizes are in 240ths of an inch.

```
#define   INCH      240
#include /usr/lib/term/terms.h

struct termtable tlp ; { \* lp is the name of the term, *\
          int bset; \* modify with new name, such as tnew *\
          int breset;
          int Hor;
          int Vert;
          int Newline;
          int Char;
          int Em;
          int Halfline;
          int Adj;
          char *twinit;
          char *twrest;
          char *twnl;
          char *hlr;
          char *hlf;
          char *flr;
          char *bdon;
          char *bdoff;
          char *iton;
          char *itoff;
          char *ploton;
          char *plotoff;
          char *up;
          char *down;
          char *right;
          char *left;
          char *codetab[256-32];
          char *zzz;
} ;
```

The meanings of the various fields are as follows:

bset bits to set in *termio.c_oflag* see **tty**(M) and **termio**(M)). after output.

breset bits to reset in *termio.c_oflag* before output.

Hor horizontal resolution in fractions of an inch.

Vert vertical resolution in fractions of an inch.

Newline space moved by a newline (linefeed) character in fractions of an inch.

Char quantum of character sizes, in fractions of an inch. (i.e., characters are multiples of Char units wide. See *codetab* below.)

Em size of an em in fractions of an inch.

Halfline space moved by a half-linefeed (or half-reverse-linefeed) character in fractions of an inch.

Adj quantum of white space for margin adjustment in the absence of the **-e** option, in fractions of an inch. (i.e., white spaces are a multiple of Adj units wide)

Note: if this is less than the size of the space character (in units of Char; see below for how the sizes of characters are defined), *nroff* will output fractional spaces using plot mode. Also, if the **-e** switch to *nroff* is used, Adj is set equal to Hor by *nroff*.

twinit set of characters used to initialize the terminal in a mode suitable for *nroff*.

twrest set of characters used to restore the terminal to normal mode.

twnl set of characters used to move down one line.

hlr set of characters used to move up one-half line.

hlf set of characters used to move down one-half line.

flr set of characters used to move up one line.

bdon set of characters used to turn on hardware boldface mode, if any. *Nroff* assumes that boldface mode is reset automatically by the *twnl* string, because many letter-quality printers reset the boldface mode when they receive a carriage return; the *twnl* string should include whatever characters are necessary to reset the boldface mode.

bdoff set of characters used to turn off hardware boldface mode, if any.

iton set of characters used to turn on hardware italics mode, if any.

itoff set of characters used to turn off hardware italics mode, if any.

ploton set of characters used to turn on hardware plot mode (for Diablo-type mechanisms), if any.

plotoff set of characters used to turn off hardware plot mode (for Diablo-type mechanisms), if any.

up set of characters used to move up one resolution unit (Vert) in plot mode, if any.

down set of characters used to move down one resolution unit (Vert) in plot mode, if any.

right set of characters used to move right one resolution unit (Hor) in plot mode, if any.

left set of characters used to move left one resolution unit (Hor) in plot mode, if any.

codetab Array of sequences to print individual characters. Order is *nroff*'s internal ordering. See the file **/usr/lib/term/tabuser.c** for the exact order.

zzz a zero terminator at the end.

The *codetab* sequences each begin with a flag byte. The top bit indicates whether the sequence should be underlined in the **.ul** font. The rest of the byte is the width of the sequence in units of *Char*.

The remainder of each *codetab* sequence is a sequence of characters to be output. Characters with the top bit off are output as given; characters with the top bit on indicate escape into plot mode. When such an escape character is encountered, *nroff* shifts into plot mode, emitting *ploton*, and skips to the next character if the escape character was '\200'.

When in plot mode, characters with the top bit off are output as given. A character with the top bit on indicates a motion. The next bit indicates coordinate, with **1** being vertical and **0** being horizontal. The next bit indicates direction, with **1** meaning up or left. The remaining five bits give the amount of the motion. An amount of zero causes exit from plot mode.

When plot mode is exited, either at the end of the string or via the amount-zero exit, *plotoff* is emitted followed by a blank.

All quantities which are in units of fractions of an inch should be expressed as INCH**num/denom*, where *num* and *denom* are respectively the numerator and denominator of the fraction; that is, 1/48 of an inch would be written as ''INCH/48''.

If any sequence of characters does not pertain to the output device, that sequence should be given as a null string.

The Development System must be installed on the computer to create a new driving table. The source code for a generic output device is in the file **/usr/lib/term/tabuser.c** Copy this file and make the necessary modifications, including the name of the termtable struct. Refer to the hardware manual for the codes needed for the output device (terminal, printer, etc.). Name the file according to the convention explained in the **term** file accompanying your *nroff* package. The makefile, **/usr/lib/term/makefile**, should be updated to include the source file to the new driving table. To perform the modification, enter the command:

```
cc -M3e -O -c tabuser.c maketerm.o -o maketerm
```

When the files are prepared, enter the command :

```
make
```

(See **make**(CP)). The source to the new driving table is linked with the object file **mkterm.o**, and the new driving table is created and installed in the proper directory.

Files

/usr/lib/term/tab*name* driving tables
/usr/lib/term/tabuser.c generic source for driving tables
/usr/lib/term/makefile makefile for creating driving tables
/usr/lib/term/mkterm.olinkable object file for creating driving tables
/usr/lib/term/terms.h used to create nroff driving tables

See Also

nroff and term in the documentation accompanying your text processing package.

Notes

Altos UNIX System V does not include *nroff*, its special **term** file, or any of the other facilities commonly associated with it. You must purchase this text processing package separately. The Development System and text processing software must be installed on the computer to create new driving tables.

Not all UNIX facilities support all of these options.

termcap

terminal capability data base

Description

The file **/etc/termcap** is a data base describing terminals. This data base is used by commands such as *vi*(C), Lyrix®, Multiplan™ and sub-routine packages such as *curses*(S). Terminals are described in *termcap* by giving a set of capabilities and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap.*

Entries in *termcap* consist of a number of fields separated by colons ':'. The first entry for each terminal gives the names that are known for the terminal, separated by vertical bars (|). For compatibility with older systems the first name is always 2 characters long. The second name given is the most common abbreviation for the terminal and the name used by *vi (C)* and *ex*(C). The last name given should be a long name fully identifying the terminal. Only the last name can contain blanks for readability.

Capabilities (including XENIX Extensions)

The following is a list of the capabilities that can be defined for a given terminal. In this list, (P) indicates padding can be specified, and (P*) indicates that padding can be based on the number of lines affected. The capability type and padding fields are described in detail in the following section "Types of Capabilities."

The codes beginning with uppercase letters (except for CC) indicate XENIX extensions. They are included in addition to the standard entries and are used by one or more application programs. As with the standard entries, not all modes are supported by all applications or terminals. Some of these entries refer to specific terminal output capabilities (such as GS for "graphics start"). Others describe character sequences sent by keys that appear on a keyboard (such as PU for PageUp key). There are also entries that are used to attribute special meanings to other keys (or combinations of keys) for use in a particular software program. Some of the XENIX extension capabilities have a similar function to standard capabilities. They are used to redefine specific keys (such as using function keys as arrow keys). The extension capabilities are included in the **/etc/termcap** file, as they are required for some utilities. The more commonly used extension capabilities are described in more detail in the section "XENIX Extensions."

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not **^H**
bs	bool		Terminal can backspace with **^H**
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
CF	str		Cursor off
ch	str	(P)	Like cm but horizontal motion only, line stays same
CL	str		Sent by CHAR LEFT key
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
CO	str		Cursor on
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like **cm**
cv	str	(P)	Like **ch** but vertical only.
CW	str		Sent by CHANGE WINDOW key
da	bool		Display may be retained above
DA	bool		Delete attribute string
db	bool		Display may be retained below
dB	num		Number of millisec of **bs** delay needed
dC	num		Number of millisec of **cr** delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of **ff** delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of **nl** delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give `:ei=:´ if **ic**
EN	str		Sent by END key
eo	bool		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
G1	str		Upper-right (1st quadrant) corner character
G2	str		Upper-left (2nd quadrant) corner character

Name	Type	Pad?	Description
G3	str		Lower-left (3rd quadrant) corner character
G4	str		Lower-right (4th quadrant) corner character
GC	str		Center graphics character (similar to ``+'')
GD	str		Down-tick character
GE	str		Graphics mode end
GG	num		Number of chars taken by GS and GE
GH	str		Horizontal bar character
GL	str		Left-tick character
GR	str		Right-tick character
GS	str		Graphics mode start
GU	str		Up-tick character
GV	str		Vertical bar character
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
HM	str		Sent by HOME key (if not **kh**)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	str		Insert mode (enter); give ':im=' if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by 'other' function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of 'keypad transmit' mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of 'other' keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in 'keypad transmit' mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on 'other' function keys
LD	str		Sent by line delete key
LF	str		Sent by line feed key
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by **vi** version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor
MP	str		Multiplan initialization string
MR	str		Multiplan reset string
ms	bool		Will scroll in stand-out mode
mu	str		Memory unlock (turn off memory lock)

Name	Type	Pad?	Description
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default \n)
ns	bool		Terminal is a CRT but doesn't scroll
NU	str		Sent by NEXT UNLOCKED CELL key
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
PD	str		Sent by PAGE DOWN key
PN	str		Start local printing
PS	str		End local printing
pt	bool		Has hardware tabs (may need to be set with **is**)
PU	str		Sent by PAGE UP key
RC	str		Sent by RECALC key
RF	str		Sent by TOGGLE REFERENCE key
RT	str		Sent by RETURN key
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by **so** or **se**
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by **us** or **ue**
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
UP	str		Sent by up-arrow key (alternate to ku)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
WL	str		Sent by WORD LEFT key
WR	str		Sent by WORD RIGHT key
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like **ce** \r \n (Delta Data)
xs	bool		Standard out not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic **so** char (Teleray 1061)

A Sample Entry

The following entry describes the Concept-100, and is among the more complex entries in the *termcap* file. (This particular Concept entry is outdated, and is used as an example only.)

```
c1 | c100 | concept100:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:\
        :cm=\Ea%+ %+ :co#80:dc=16\E^A:dl=3*\E^B:\
        :ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue over to multiple lines by giving a backslash (\) as the last character of a line. Empty fields can be included for readability between the last field on a line and the first field on the next. Capabilities in *termcap* are of three types: Boolean capabilities, which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence that can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has 'automatic margins' (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. The description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the rest of the string is sent to provide this delay. The delay can be either a integer, e.g., '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a colon (:) in a capability, it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability, it must be encoded as **\200**. The routines that deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it. To test a new terminal description, you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in **/etc/termcap.** TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor.

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, it should have the **am** capability. If the terminal can clear its screen, this is given by the **cl** string capability. If the terminal can backspace, it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H** in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over), it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on (i.e., **am**).

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as:

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability. This capability uses *printf*(S) like escapes (such as **%x**) in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, its arguments are the line and then the column to which motion is desired, and the **%** encodings have the following meanings:

%d	replaced by line/column position, 0 origin
%2	like %2d - 2 digit field
%3	like %3d - 3 digit field
%.	like *printf*(S) %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output
%r	reverses order of line and column, no output
%i	increments line/column position (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output
%D	Reverse coding (x-2*(x%16)), no output (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is 'cm=6\E&%r%2c%2Y'. The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, 'cm=^T%.%.'. Terminals that use '%.' need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus 'cm=\E=%+ %+ '.

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, it should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen), this can be given as **ho**; similarly, a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, the sequence should be given as **ce**. If the terminal can clear from the current position to the end of the display, the sequence should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, the sequence should be given as **al**. Note that this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line on which the cursor rests, the sequence should be given as **dl**. This is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, the sequence can be given as **sb**, but **al** can suffice. If the terminal can retain display memory above, the **da** capability should be given, and if display memory can be retained below, then **db** should be given. These let the editor know that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to the insert/delete character that can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and entering text separated by cursor motions. Enter 'abc def', using local cursor motions (not spaces) between the 'abc' and the 'def'. Then position the cursor before the 'abc' and put the terminal in insert mode. If entering characters causes the rest of the line to shift rigidly and characters to fall off the end, your terminal does not distinguish between blanks and untyped positions. If the 'abc' shifts over to the 'def' which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for 'insert null'. No known terminals have an insert mode, not falling into one of these two classes.

The editor can handle both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Specify **im** as the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Specify **ei** as the sequence to leave insert mode (specify this with an empty value if you also gave **im** an empty value). Now specify **ic** as any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not support **ic**, terminals that send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode, these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as reverse video, blinking, or underlining - half bright is not usually an acceptable 'standout' mode unless the terminal is in reverse video mode constantly), the preferred mode is reverse video by itself. It is acceptable, if the code to change into or out of standout mode leaves one, or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do. Although it may confuse some programs slightly, it cannot be helped.

Codes to begin underlining and end underlining can be given as **us**, and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, the sequence can be given as **uc**. (If the underline code does not move the cursor to the right, specify the code followed by a nondestructive space.)

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), the sequence can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, the sequence can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the

Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed), even though it does not overstrike, you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by specifying **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not to transmit, enter these codes as **ks** and **ke**. Otherwise, the keypad is assumed always to transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh**. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, ':ko=cl,ll,sf,sb:', which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete, but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the Mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, the sequence can be given as **ta**.

Terminals that do not allow '~' characters to be displayed (such as Hazeltines), should indicate **hz**. Datamedia terminals that echo carriage-return-linefeed for carriage return, and then ignore a following linefeed, should indicate **nc**. Early Concept terminals, that ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

If the leading character for commands to the terminal (normally the escape character) can be set by the software, specify the command character(s) with the capability **CC**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** is displayed before **if**. This is useful where **if** is **/usr/lib/tabset/std** , but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability, **tc**, can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since *termlib* routines search the entry from left to right, and since the **tc** capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with **xx@** where xx is the capability. For example:

```
hn | 2621nl:ks@:ke@:tc=2621:
```

This defines a 2621nl that does not have the **ks** or **ke** capabilities, and does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

XENIX Extensions

Capabilities This table lists the (previously listed) XENIX extensions to the termcap capabilities. It shows which codes generate information input from the keyboard to the program reading the keyboard and which codes generate information output from the program to the screen.

Name	Input/Output	Description
CF	str	Cursor off
CL	str	Sent by CHAR LEFT key
CO	str	Cursor on
CW	str	Sent by CHANGE WINDOW key
DA	bool	Delete attribute string
EN	str	Sent by END key
G1	str	Upper-right (1st quadrant) corner character
G2	str	Upper-left (2nd quadrant) corner character
G3	str	Lower-left (3rd quadrant) corner character
G4	str	Lower-right (4th quadrant) corner character
G5	str	Upper right (1st quadrant) corner character (double)
G6	str	Upper left (2nd quadrant) corner character (double)
G7	str	Lower left (3rd quadrant) corner character (double)
G8	str	Lower right (4th quadrant) corner character (double)
GC	str	Center graphics character (similar to +)
Gc	str	Centre graphics character (double)
GD	str	Down-tick character
Gd	str	Down tick character (double)
GE	str	Graphics mode end
GG	num	Number of chars taken by GS and GE
GH	str	Horizontal bar character
Gh	str	Horizontal bar character (double)
GL	str	Left-tick character
Gl	str	left-tick character (double)
GR	str	Right-tick character
Gr	str	right-tick character (double)
GS	str	Graphics mode start
GU	str	Up-tick character
Gu	str	Up-tick character (double)
GV	str	Vertical bar character
Gv	str	Vertical bar character (double)
HM	str	Sent by HOME key (if not **kh**)
mb	str	blinking on
me	str	blinking off
MP	str	Multiplan initialization string
MR	str	Multiplan reset string
NU	str	Sent by NEXT UNLOCKED CELL key
PD	str	Sent by PAGE DOWN key
PU	str	Sent by PAGE UP key
RC	str	Sent by RECALC key
RF	str	Sent by TOGGLE REFERENCE key
RT	str	Sent by RETURN key
UP	str	Sent by up-arrow key (alternate to ku)
WL	str	Sent by WORD LEFT key
WR	str	Sent by WORD RIGHT key

Cursor motion Some application programs make use of special editing codes. **CR** and **CL** move the cursor one character right and left respectively. **WR** and **WL** move the cursor one word right and left respectively. **CW** changes windows, when they are used in the program.

Some application programs turn off the cursor. This is accomplished using **CF** for cursor off and **CO** to turn it back on.

Graphic mode. If the terminal has graphics capabilities, this mode can be turned on and off with the **GS** and **GE** codes. Some terminals generate graphics characters from all keys when in graphics mode (such as the Visual 50). The other **G** codes specify particular graphics characters accessed by escape sequences. These characters are available on some terminals as alternate graphics character sets (not as a bit-map graphic mode). The vt100 has access to this kind of alternate graphics character set, but not to a bit-map graphic mode.

Files

/etc/termcap File containing terminal descriptions

See Also

ex(C), curses(S), termcap(S), tset(C), vi(C), more(C), screen(HW)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

ex(C) allows only 256 characters for string capabilities, and the routines in *termcap(S)* do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the *vi*(C) program.

Not all programs support all entries. There are entries that are not supported by any program.

XENIX termcap extensions are explained in detail in the software application documentation.

Refer to the *screen*(HW) manual page, for a description of the character sequences used by the monitor device on your specific system.

terminfo

format of compiled terminfo file

Description

Compiled terminfo descriptions are placed under the directory **/usr/lib/terminfo**. In order to avoid a linear search of a huge UNIX system directory, a two-level scheme is used: **/usr/lib/terminfo/c/name** where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, *act4* can be found in the file **/usr/lib/terminfo/a/act4**. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8- or more-bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the *tic*(C) program, and read by the routine *setupterm* in *terminfo*(S). The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal 0432); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value -1 is represented by 0377, 0377; other negative values are illegal. The -1 generally means that a capability is missing from this terminal. Note that this format corresponds to the hardware of the VAX and PDP-11. Machines in which this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the '|' character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1, as the flag is present or absent. The capabilities are in the same order as the file **<term.h>**.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short-word boundary.

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is -1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of -1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information $<nn> and parameter information %x are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null-terminated.

Note that it is possible for *setupterm* to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since *setupterm* was recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine *setupterm* must be prepared for both possibilities; this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
    cr=^M, cud1=^J, ind=^J, bel=^G, am, cub1=^H,
    ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
    cols#80, lines#24, cuf1=^X, cuu1=^Z, home=^],
```

```
3000 032 001      \0 025  \0  \b  \0 212  \0   "  \0   m   i   c   r
020    o   t   e   r   m   |   a   c   t   4   |   m   i   c   r   o
040    t   e   r   m       a   c   t       i   v  \0  \0 001  \0  \0
060   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
100   \0  \0   P  \0 377 377 030  \0 377 377 377 377 377 377 377 377
120  377 377 377 377  \0  \0 002  \0 377 377 377 377 004  \0 006  \0
140   \b  \0 377 377 377 377  \n  \0 026  \0 030  \0 377 377 032  \0
160  377 377 377 377 034  \0 377 377 036  \0 377 377 377 377 377 377
200  377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520  377 377 377 377      \0 377 377 377 377 377 377 377 377 377 377
```

```
540 377 377 377 377 377 377 007  \0  \r  \0  \f  \0 036  \0 037  \0
560 024   %   p   1   %   c   %   p   2   %   c  \0  \n  \0 035  \0
600  \b  \0 030  \0 032  \0  \n  \0
```

Some limitations: the total size of a compiled description cannot exceed 4096 bytes; the name field cannot exceed 128 bytes.

Files

/usr/lib/terminfo/*/* compiled terminal capability data base

See Also

terminfo(M), terminfo(S), tic(C)

timezone

set default system time zone

Syntax

/etc/TIMEZONE

Description

This file sets and exports the time zone environmental variable **TZ**.

This file is "dotted" into other files that must know the time zone, including **/etc/cshrc, /etc/profile, /etc/rc2, .profile.**

TZ contains the following information:

(*sss*) One to nine letters designating the standard time zone.

(*n*) Number of hours past Greenwich mean time for the standard time (partial hours are valid e.g. 12:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich.

(*ddd*) One to nine letters designating the local daylight savings time (summer time) zone. If not present, summer time is assumed not to apply.

(*m*) Number of hours past Greenwich mean time for the summer time (partial hours are valid e.g. 11:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich. If *m* is not given, the distance to GMT during summer time is assumed to be one hour less than during standard time.

(*start*) The rule defining the day summer time begins. In the southern hemisphere, the ending day will be earlier in the year than the starting day.

(*end*) The rule defining the day summer time ends.

(*time*) The time of day the change to and from summer time occurs. The default is 02:00:00 local time.

The rules for defining the **start** and **end** of summer time are as follows:

J*n*	1 based Julian day *n* ($1 \le n \le 365$)*
n	0 based Julian day *n* ($0 \le n \le 364$)*
W*n.d*	day *d* ($0 \le d \le 6$)** of week *n* ($1 \le n \le 53$)†
M*m.n.d*	day *d* of week *n* ($1 \le n \le 5$)‡ of month *m* ($1 \le m \le 12$)

* Leap days (February 29) are never counted; that is, February 28 (J59) is immediately followed by March 1 (J60) even in leap years.

** Sunday is the first day of the week (0). If *d* is omitted, Sunday is assumed. Note that *d* is optional.

† The 5th week of the month is always the last week containing day *d*, whether there are actually 4 or 5 weeks containing day *d*.

‡ The 53rd week of the year is always the last week containing day *d*, whether there are actually 52 or 53 weeks containing day *d*.

If **start** and **end** are omitted, current U.S. law is assumed.

Examples

A simple setting for New Jersey could be

```
TZ='EST5EDT'
```

where **EST** is the abbreviation for the main time zone, **5** is the difference, in hours, between GMT (Greenwich Mean Time) and the main time zone, and **EDT** is the abbreviation for the alternate time zone.

The most complex representation of the same setting, for the year 1986, is

```
TZ='EST5:00:00EDT4:00:00;117/2:00:00,299/2:00:00'
```

where **EST** is the abbreviation for the main time zone, **5:00:00** is the difference, in hours, minutes, and seconds between GMT and the main time zone, **EDT** is the abbreviation for the alternate time zone, **4:00:00** is the difference, in hours, minutes, and seconds between GMT and the alternate time zone, **117** is the number of the day of the year (Julian day) when the alternate time zone will take effect, **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will take effect, **299** is the number of the day of the year when the alternate time zone will end, and **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will end.

A southern hemisphere setting such as the Cook Islands could be

TZ='KDT9:30KST10:00;64/5:00,303/20:00'

This setting means that **KDT** is the abbreviation for the main time zone, **KST** is the abbreviation for the alternate time zone, KST is **9** hours and **30** minutes later than GMT, KDT is **10** hours later than GMT, the starting date of KDT is the **64**th day at **5** AM, and the ending date of KDT is the **303**rd day at **8** PM.

Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be midnight.

Note that in most installations, **TZ** is set to the correct value by default when the user logs on, via the local */etc/profile* file [see *profile*(F)].

See Also

ctime(S), profile(F), environ(M), TZ(M), rc2(ADM)

Notes

Setting the time during the interval of change from the main time zone to the alternate time zone or vice versa can produce unpredictable results.

Standards Conformance

timezone is conformant with:

The X/Open Portability Guide II of January 1987.

top, top.next

the Micnet topology files

Description

These files contain the topology information for a Micnet network. The topology information describes how the individual systems in the network are connected, and what path a message must take from one system to reach another. Each file contains one or more lines of text. Each line of text defines a connection or a communication path.

The **top** file defines connections between systems. Each line lists the machine names of the connected systems, the serial lines used to make the connection, and the speed (baud rate) of transmission between the systems. Each line has the following format:

machine1 tty1a machine2 tty2a speed

machine1 and *machine2a* are the machine names of the respective systems (as given in the **systemid** files). The *ttys* are the device names (e.g., tty1a) of the connecting serial lines. The speed must be an acceptable baud rate (e.g., 110, 300, ..., 19200).

The **top.next** file contains information about how to reach a particular system from a given system. There may be several lines for each system in the network. Each line lists the machine name of a system, followed by the machine name of a system connected to it, followed by the machine names of all the systems that may be reached by going through the second system. Such a line has the form:

machine1 machine2 machine3 [machine4]...

The machine names must be the names of the respective systems (as given by the first machine name in the **systemid** files).

The *top.next* file must be present even if there are only two computers in the network. In such a case, the file must be empty.

In the **top** and **top.next** files, any line beginning with a number sign (#) is considered a comment, and is ignored.

Files

/usr/lib/mail/top

/usr/lib/mail/top.next

See Also

netutil(ADM), systemid(F)

types

primitive system data types

Syntax

#include <sys/types.h>

Description

The data types defined in the include file **<sys/types.h>** are used in UNIX system code; some data of these types are accessible to user code.

The form *daddr_t* is used for disk addresses except in an inode on disk, see *filesystem* (F). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

See Also

filesystem(F)

Standards Conformance

types is conformant with:

The X/Open Portability Guide II of January 1987.

unistd

file header for symbolic constants

Syntax

#include <unistd.h>

Description

The header file **<unistd.h>** lists the symbolic constants and structures not already defined or declared in some other header file.

```
/* Symbolic constants for the "access" routine: */

#define R_OK      4    /*Test for Read permission */
#define W_OK      2    /*Test for Write permission */
#define X_OK      1    /*Test for eXecute permission */
#define F_OK      0    /*Test for existence of File */

#define F_ULOCK   0    /*Unlock a previously locked region */
#define F_LOCK    1    /*Lock a region for exclusive use */
#define F_TLOCK   2    /*Test and lock a region for exclusive use */
#define F_TEST    3    /*Test a region for other processes locks */

/*Symbolic constants for the "lseek" routine: */

#define SEEK_SET  0    /* Set file pointer to "offset" */
#define SEEK_CUR  1    /* Set file pointer to current plus "offset" */
#define SEEK_END  2    /* Set file pointer to EOF plus "offset" */

/*Path names:*/

#define GF_PATH   "/etc/group"    /*Path name of the group file */
#define PF_PATH   "/etc/passwd"   /*Path name of the passwd file */
```

Standards Conformance

unistd is conformant with:

The X/Open Portability Guide II of January 1987.

utmp, wtmp

formats of utmp and wtmp entries

Syntax

#include <sys/types.h>
#include <utmp.h>

Description

These files, which hold user and accounting information for such commands as *who*(C), *write*(C), and *login*(M), have the following structure as defined by **<utmp.h>**:

```
#define   UTMP_FILE    ''/etc/utmp''
#define   WTMP_FILE    ''/etc/wtmp''
#define   ut_name      ut_user

struct utmp {
       char        ut_user[8];        /* User login name */
       char        ut_id[4];          /* usually line # */
       char        ut_line[12];       /* device name (console, lnxx) */
       short       ut_pid;            /* process id */
       short       ut_type;           /* type of entry */
       struct      exit_status {
           short       e_termination; /* Process termination status */
           short       e_exit;        /* Process exit status */
       } ut_exit;                     /* The exit status of a process
                                         marked as DEAD_PROCESS. */
       time_t      ut_time;           /* time entry was made */
};

/*  Definitions for ut_type  */

#define EMPTY              0
#define RUN_LVL                1
#define BOOT_TIME          2
#define OLD_TIME           3
#define NEW_TIME           4
#define INIT_PROCESS       5   /* Process spawned by "init" */
#define LOGIN_PROCESS    6 /* A "getty" process waiting for login */
#define USER_PROCESS       7   /* A user process */
#define DEAD_PROCESS       8
#define ACCOUNTING         9
#define UTMAXTYPE ACCOUNTING /* Largest legal value of ut_type */
```

```
/*  Special strings or formats used in the "ut_line" field when  */
/*  accounting for something other than a process  */
/*  No string for the ut_line field can be more than 11 chars +  */
/*  a NULL in length  */
#define RUNLVL_MSG "run-level %c"
#define BOOT_MSG   "system boot"
#define OTIME_MSG  "old time"
#define NTIME_MSG  "new time"
```

Files

/usr/include/utmp.h
/etc/utmp
/etc/wtmp

See Also

getut(S), login(M), who(C), write(C)

Standards Conformance

utmp and *wtmp* are conformant with:

The X/Open Portability Guide II of January 1987.

x.out

format of XENIX link editor output

Syntax

#include <x.out.h>

Description

The output of the XENIX link editor, called the *x.out* or segmented *x.out* format, is defined by the files **/usr/include/x.out.h** and **/usr/include/sys/relsym.h**. The **x.out** file has the following general layout:

1. Header.

2. Extended header.

3. File segment table (for segmented formats).

4. Segments (Text, Data, Symbol, and Relocation).

In the segmented format, there may be several text and data segments, depending on the memory model of the program. Segments within the file begin on boundaries which are multiplies of 512 bytes as defined by the file's pagesize.

Format

```
/*
 * The main and extended header structures.
 * For x.out segmented (XE_SEG):
 *     1) fields marked with (s) must contain sums of xs_psize for
 *     non-memory images, or xs_vsize for memory images.
 *     2) the contents of fields marked with (u) are undefined.
 */

struct xexec {               /* x.out header */
   unsigned short x_magic;  /* magic number */
   unsigned short x_ext;    /* size of header extension */
   long     x_text;     /* size of text segment (s) */
   long     x_data;     /* size of initialized data (s) */
   long     x_bss;   /* size of uninitialized data (s) */
   long     x_syms;     /* size of symbol table (s) */
   long     x_reloc; /* relocation table length (s) */
   long     x_entry; /* entry point, machine dependent */
```

```
    char      x_cpu;    /* cpu type & byte/word order */
    char      x_relsym; /* relocation & symbol format (u) */
    unsigned short  x_renv;      /* run-time environment */
};


struct xext {                   /* x.out header extension */
    long      xe_trsize;        /* size of text relocation (s) */
    long      xe_drsize;        /* size of data relocation (s) */
    long      xe_tbase;     /* text relocation base (u) */
    long      xe_dbase;     /* data relocation base (u) */
    long      xe_stksize;       /* stack size (if XE_FS set) */
              /* the following must be present if XE_SEG */
    long      xe_segpos;        /* segment table position */
    long      xe_segsize;       /* segment table size */
    long      xe_mdtpos;    /* machine dependent table position */
    long      xe_mdtsize;   /* machine dependent table size */
    char      xe_mdttype;   /* machine dependent table type */
    char      xe_pagesize;  /* file pagesize, in multiples of 512 */
    char      xe_ostype;        /* operating system type */
    char      xe_osvers;        /* operating system version */
    unsigned short  xe_eseg;  /* entry segment, machine dependent */
    unsigned short  xe_sres;  /* reserved */
};


struct xseg {                 /* x.out segment table entry */
    unsigned short  xs_type;  /* segment type */
    unsigned short  xs_attr;  /* segment attributes */
    unsigned short  xs_seg;       /* segment number */
    char      xs_align;     /* log base 2 of alignment */
    char      xs_cres;      /* unused */
    long      xs_filpos;        /* file position */
    long      xs_psize;     /* physical size (in file) */
    long      xs_vsize;     /* virtual size (in core) */
    long      xs_rbase;     /* relocation base address/offset */
    unsigned short  xs_noff;  /* segment name string table offset */
    unsigned short  xs_sres;  /* unused */
    long      xs_lres;      /* unused */
};


struct xiter {                /* x.out iteration record */
    long      xi_size;      /* source byte count */
    long      xi_rep;       /* replication count */
    long      xi_offset;    /* destination offset in segment */
};

struct xlist {                /* xlist structure for xlist(3). */
    unsigned short  xl_type;  /* symbol type */
    unsigned short  xl_seg;       /* file segment table index */
    long      xl_value;       /* symbol value */
    char      *xl_name;       /* pointer to asciz name */
};

struct aexec {                /* a.out header */
```

```
    unsigned short  xa_magic;     /* magic number */
    unsigned short  xa_text;      /* size of text segment */
    unsigned short  xa_data;      /* size of initialized data */
    unsigned short  xa_bss;       /* size of uninitialized data */
    unsigned short  xa_syms;      /* size of symbol table */
    unsigned short  xa_entry;     /* entry point */
    unsigned short  xa_unused;    /* not used */
    unsigned short  xa_flag;      /* relocation info stripped */
};


struct nlist {                /* nlist structure for nlist(3). */
    char       n_name[8];     /* symbol name */
    int    n_type;            /* type flag */
    unsigned n_value;         /* value */
};


struct bexec {              /* b.out header */
    long   xb_magic; /* magic number */
    long   xb_text;  /* text segment size */
    long   xb_data;  /* data segment size */
    long   xb_bss;   /* bss size */
    long   xb_syms;  /* symbol table size */
    long   xb_trsize;   /* text relocation table size */
    long   xb_drsize;   /* data relocation table size */
    long   xb_entry; /* entry point */
};
```

See Also

masm(CP), ld(CP), nm(CP), strip(CP), xlist(S)

Value Added

x.out is an extension of AT&T System V provided in Altos UNIX System V.

xbackup

XENIX incremental dump tape format

Description

The *xbackup* and *xrestore* commands are used to write and read incremental dump magnetic tapes.

The backup tape consists of a header record, some bit mask records, a group of records describing file system directories, a group of records describing file system files, and some records describing a second bit mask.

The header record and the first record of each description have the format described by the structure included by:

#include <dumprestor.h>

Fields in the *dumprestor* structure are described below.

NTREC is the number of 512 byte blocks in a physical tape record. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE	Tape volume label.
TS_INODE	A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.
TS_BITS	A bit mask follows. This bit mask has one bit for each inode that was backed up.
TS_ADDR	A subblock to a file (*TS_INODE*). See the description of *c_count* below.
TS_END	End of tape record.
TS_CLRI	A bit mask follows. This bit mask contains one bit for all inodes that were empty on the file system when backed up.
MAGIC	All header blocks have this number in *c_magic*.

CHECKSUM	Header blocks checksum to this value.

The fields of the header structure are as follows:

c_type The type of the header.

c_date The date the backup was taken.

c_ddate The date the file system was backed up.

c_volume The current volume number of the backup.

c_tapea The current block number of this record. This is counting 512 byte blocks.

c_inumber The number of the inode being backed up if this is of type TS_INODE.

c_magic This contains the value MAGIC above, truncated as needed.

c_checksum This contains whatever value is needed to make the block sum to CHECKSUM.

c_dinode This is a copy of the inode as it appears on the file system.

c_count The following count of characters describes the file. A character is zero if the block associated with that character was not present on the file system; otherwise, the character is nonzero. If the block was not present on the file system no block was backed up and it is replaced as a hole in the file. If there is not sufficient space in this block to describe all of the blocks in a file, TS_ADDR blocks will be scattered through the file, each one picking up where the last left off.

c_addr This is the array of characters that is used as described above.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END block and then the tapemark.

The structure *idates* describes an entry of the file where backup history is kept.

See Also

xbackup(ADM), xrestore(ADM), filesystem(F)

Value Added

xbackup is an extension of AT&T System V provided in Altos UNIX System V.

xprtab

system tty transparent printer map file

Description

The **xprtab** file is used by *xprsetup*(ADM) to map transparent printer device nodes to tty device nodes. The **xprtab** file is normally maintained and updated using *pcu*(ADM), the Altos port configuration utility. If the **xprtab** file is changed manually, the transparent printer tty mapping change will not take place until *xprsetup*(ADM) is executed.

Each entry in the **xprtab** file contains three fields separated by white space:

xprnum *ttypathname* *termtype*

Each field is described below:

xprnum This field contains a unique decimal number in the range **1** through **99**. This number represents the minor device number of the transparent printer device node. Transparent printer device node names range from **/dev/xpr/xpr01** to **/dev/xpr/xpr99**.

ttypathname This field contains the absolute pathname of the associated tty device node (e.g., **/dev/tty1a**).

termtype This field contains the terminal *terminfo*(M) name of the terminal connected to the tty port in field two. For tty ports connected to modems this field should contain the termtype **dialup**.

Files

/etc/xprtab

See Also

xprsetup(ADM), pcu(ADM)

Value Added

xprtab is an extension to AT&T UNIX System V provided in Altos UNIX System V.

Permuted Index

Commands, System Calls, Library Routines and File Formats

This permuted index is derived from the "Name" description lines found on each reference manual page. Each *index* line shows the title of the entry to which the line refers, followed by the reference manual section letter where the page is found.

To use the *permuted index* search the middle column for a key word or phrase. The right hand column contains the name and section letter of the manual page that documents the key word or phrase. The left column contains additional useful information about the command. Commands or routines are also listed in the context of the *index* line, followed by a colon (:). This denotes the "beginning" of the sentence. Notice that in many cases, the lines wrap, starting in the middle column and ending in the left column. A slash (/) indicates that the description line is truncated.

Change Information

CH

This is a summary of the changes that have been made to the previous version of this manual. The chapters, page numbers, and/or paragraphs mentioned in this summary refer to the previous manual.

Title: Altos UNIX System V/386 Release 3.2
User's Reference (C,M,F)

Revised Part Number: 690-23414-002

Previous Part Numbers: 690-23414-001 and 690-24536-001

Date: March 1991

Changes:

The phrase, "for the 486" (or in some manuals "for Entry-level Systems") was deleted to indicate that this operating system runs on a wider range of platforms.

The "Guide to Your Altos UNIX System V/386 Release 3.2 Documentation" and the "Operating System Documents for Different Audiences" pages located in the front matter of the manual were both changed to indicate that a different combination of the available manuals are now shipped with every run-time system. This has also affected which manuals are now available as spare parts.

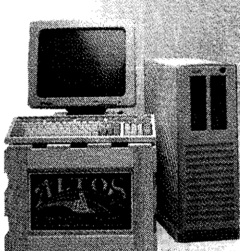
Added *fuser*(C), *hostid*(C), *ksh*(C), *otar*(C), *pscat*(C), *rcvtrip*(C), *setmode*(C), *xprcat*(C), *isverify*(M), *powerfail*(M), *restart*(M), *card_info*(F), *hs*(F), *maildelivery*(F), *meisa*(F), *memtune*(F), and *xprtab*(F).

Deleted *ct*(C).

Miscellaneous editing and typographical changes were made throughout the manual.

Page	Command	Description
1	cpio(C)	Added **-T** and **-K** options.
1	date(C)	Added **-c**, **-m**, and **-s** options.
3	dd(C)	Corrected block size used in example, from 1 K to 512-byte blocks
1	devnm(C)	Corrected disk device reference, from **hd1** to **hdb**.
1	dos(C)	Inserted description of *dos*(C) utilities, which was mistakenly omitted from some versions of the document.
6	dos(C)	Corrected disk device references, from **rhd0d** and **rhd1d** to **rhdad** and **rhdbd** respectively.
1	echo(C)	Added **-n** option. Removed the note detailing differneces between *csh* built-in *echo* and */bin/echo*. Added a note describing how to echo octal characters. Removed ''Standards Conformance'' section.
1, 5	man(C)	Deleted CT designator references, and emphasized that *nroff* text processing package must be installed separately.
1, 2	more(C)	Deleted **-r** option.
1	newform(C)	Added more information clarifying the use of the **-l** option.
1-5	passwd(C)	Substantially revised the description of *passwd*. This included changing valid options and arguments, and adding new settings to **/etc/default/passwd**. Also, removed the ''Standards Conformance'' section.
2	ps(C)	Deleted **-s** option, for specifying an alternate swap file.

P/N 690-23414-002
Printed in U.S.A.
8/91



Altos Computer Systems
2641 Orchard Parkway, San Jose, CA 95134
408/432-6200, FAX 408/435-8517