PLEXUS

Sys3 UNIX Programmer's Manual -- vol 1A

98-05045.8 Rev A September 24, 1984

PLEXUS

Sys3 UNIX vol 1A Programmer's Manual

Sys3 UNIX Programmer's Manual -- vol 1A

98-05045.8 Rev A September 24, 1984

PLEXUS COMPUTERS, INC.

3833 North First St.

San Jose, CA 95134

408/943-9433

Printed in the United States of America

REVISION RECORD

Plexus Sys3 UNIX Programmer's Manual -- vol 1A

REVISION LEVEL	DATE	DESCRIPTION
98-05046.1	December 23, 1982	First edition
98-05046.2	January 15, 1983	Editorial changes
98-05046.3	May 6, 1983	Removed unsupported graphics utilities; other small editorial changes
98-05046.4	June 20, 1983	Changes for Sys3 Rev. 3.0 (MC68000 version)
98-05046.5	August 19, 1983	New pages for the Plexus Network Operating System (NOS); other small additions and corrections
98-05046.6	November 18, 1983	Several new pages; other small additions and corrections
98-05046.7	April 6, 1984	First typeset version; additions and corrections
98-05046.8	September 24, 1984	Commands to support new LP spooler; additions and corrections

ACKNOWLEDGEMENTS

The form and much of the content of this manual come from the *UNIX Programmer's Manual Release 3.0* (Volume 1), edited by T. A. Dolotta, S. B. Olsson, and A. G. Petruccelli.

PLEXUS INTRODUCTION

This release of the *Plexus Sys3 UNIX Programmer's Manual* is designed for use with Plexus Sys3. This manual includes a number of commands that are not part of stock SYSTEM III, plus enhancements to SYSTEM III commands. The majority of these are in Section 1 ("Commands and Application Programs"). Therefore, Volume 1 was separated into two different physical volumes. Section 1 is now in physical Volume 1A, and Sections 2 through 8 are in Volume 1B.

Some SYSTEM III commands are designed for use with UNIX systems on specific hardware such as the PDP-11; these commands are inappropriate for use on Plexus systems, and are thus not supported by Plexus. No source was provided for other SYSTEM III commands. The following table lists all the SYSTEM III commands that are not supported by Plexus, along with codes indicating why Plexus does not support them. The codes have the following meanings:

NA - Applicable to other hardware.
NI - Not implemented.
NS - No source available.

Command	Function	Code
as.pdp	assembler for PDP-11	NA
as.vax	assembler for VAX-11/780	NA
chess	the game of chess	NS
dj	DJ-11 asynchronous multiplexor	NA
dmc	communications link with built-in DDCMP protocol	NA
dn	DN-11 ACU interface	NA
dpr	off-line print	NA
dqs	DQS-11 interface for two-point BSC	NA
du	DU-11 synchronous line interface	NA
dz	DZ-11, DZ-11/KMC-11, DH-11 asynchronous multiplexors	NA
etp	Equipment Test Package	NA
fget	retrieve files from the HONEYWELL 6000	NA
fget.demon	file retrieval daemons	NA
fptrap	floating point interpreter	NA
fscv	convert files between PDP-11 and VAX-11/780 systems	NA
fsend	send files to the HONEYWELL 6000	NA
gcat	send phototypesetter output to the HONEYWELL 6000	NA
gcosmail	send mail to HIS user	NA
gdev	graphical device routines and filters	NI
ged	graphical editor	NI
gps	format of graphical files	NI
graphics	access graphical and numerical commands	NI
gutil	graphical utilities	NI
hasp	RJE (Remote Job Entry) to IBM	NA
hp	RP04/RP05/RP06 moving-head disk	NA
hs	RH11/RJS03-RJS04 fixed-head disk file	NA
ht	TU16 magnetic tape interface	NA
kas	assembler for the KMC11 microprocessor	NA
kl	KL-11 or DL-11 asynchronous interface	NA

kmc	KMC11 microprocessor	NA
kun	un-assembler for the KMC11/DMC11 microprocessor	NA
maze	generate a maze	NS
pcl	parallel communications link interface	NA
reversi	a game of dramatic reversals	NS
rf	RF11/RS11 fixed-head disk file	NA
rk	RK-11/RK03 or RK05 disk	NA
rl	RL-11/RL01 disk	NA
rp	RP-11/RP03 moving-head disk	NA
sdb	symbolic debugger	NA
sky	obtain ephemerides	NS
st	synchronous terminal interface	NA
stat	statistical network for graphical commands	NI
tm	TM11/TU10 magnetic tape interface	NA
toc	graphical table of contents routines	NI
vaxops	VAX-11/780 console operations	NA
vlx	VAX-11/780 LSI console floppy interface	NA

See the Introductions to each section for information on new commands.

BELL INTRODUCTION

(This Introduction was written by Bell Laboratories for the *UNIX User's Manual Release 1.0.*)

This manual describes the features of UNIX. It provides neither a general overview of UNIX (for that, see "The UNIX Time-Sharing System," *BSTJ*, Vol. 57, No. 6, Part 2, pp. 1905-29, by D. M. Ritchie and K. Thompson), nor details of the implementation of the system (see "UNIX Implementation," *BSTJ*, same issue, pp. 1931-46).

Not all commands, features, and facilities described in this manual are available in every UNIX system; for example, *yacc*(1) is usually not available in a UNIX system running on a PDP-11/23. When in doubt, consult your system's administrator.

This manual is divided into eight sections, some containing inter-filed sub-classes:

1. Commands and Application Programs:
   1. General-Purpose Commands.
   1C. Communications Commands.
   1G. Graphics Commands.
   1M. System Maintenance Commands.
2. System Calls.
3. Subroutines:
   3C. C and Assembler Library Routines.
   3M. Mathematical Library Routines.
   3S. Standard I/O Library Routines.
   3X. Miscellaneous Routines.
4. Special Files.
5. File Formats.
6. Games.
7. Miscellaneous Facilities.
8. System Maintenance Procedures.

Section 1 (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory **/bin** (for **bin**ary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the *shell*. Sub-class 1C contains communication programs such as *cu*, *dpr*, etc. These entries may differ from system to system. Sub-class 1M contains system maintenance programs such as *fsck*, *mkfs*, etc., which generally reside in the directory **/etc**; these commands are not intended for use by the ordinary user due to their privileged nature. Some UNIX systems have a directory called **/usr/lbin**, containing local commands.

Section 2 (*System Calls*) describes the entries into the UNIX supervisor, including the C language interface.

Section 3 (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4 (*Special Files*) discusses the characteristics of each system file that actually refers to an input/output device. The names in this section generally refer to the Digital Equipment Corporation's device names for the hardware, rather than to the names of the special files themselves.

Section 5 (*File Formats*) documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(5). Excluded are files used by only one

command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories **/usr/include** and **/usr/include/sys**.

Section 6 (*Games*) describes the games and educational programs that, as a rule, reside in the directory **/usr/games**.

Section 7 (*Miscellaneous Facilities*) contains a variety of things. Included are descriptions of character sets, macro packages, etc.

Section 8 (*System Maintenance Procedures*) discusses crash recovery and boot procedures, etc. Information in this section is not of great interest to most users.

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

The **NAME** part gives the name(s) of the entry and briefly states its purpose.

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Commands*):

Boldface strings are literals and are to be typed just as they appear.

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).

Square brackets **[]** around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Ellipses **...** are used to show that the previous argument prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus —, plus +, or equal sign = is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with —, +, or =.

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

The **FILES** part gives the file names that are built into the program.

The **SEE ALSO** part gives pointers to related information.

The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The **WARNINGS** part points out potential pitfalls.

The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permuted index derived from that table precede Section 1. On each *index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call. On most systems, all entries are available on-line via the *man*(1) command, q.v.

HOW TO GET STARTED

This discussion provides the basic information you need to get started on UNIX: how to log in and log out, how to communicate through your terminal, and how to run a program. (See *UNIX for Beginners* by B. W. Kernighan for a more complete introduction to the system.)

Logging in. You must dial up UNIX from an appropriate terminal. UNIX supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained (together with the telephone number(s) of your UNIX system) from the administrator of your system. Common terminal speeds are 10, 15, 30, and 120 characters per second (110, 150, 300, and 1,200 baud); occasionally, speeds of 240, 480, and 960 characters per second (2,400, 4,800, and 9,600 baud) are also available. On some UNIX systems, there are separate telephone numbers for each available terminal speed, while on other systems several speeds may be served by a single telephone number. In the latter case, there is one "preferred" speed; if you dial in from a terminal set to a different speed, you will be greeted by a string of meaningless characters (the **login:** message at the wrong speed). Keep hitting the "break" or "attention" key until the **login:** message appears. Hard-wired terminals usually are set to the correct speed.

Most terminals have a speed switch that should be set to the appropriate speed and a half-/full-duplex switch that should be set to full-duplex. When a connection (at the speed of the terminal) has been established, the system types **login:** and you then type your user name followed by the "return" key. If you have a password (and you should!), the system asks for it, but does not print ("echo") it on the terminal. After you have logged in, the "return", "new-line", and "line-feed" keys will give exactly the same result.

It is important that you type your login name in lower case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and that you mean all subsequent upper-case input to be treated as lower case. When you have logged in successfully, the shell will type a **$** to you. (The shell is described below under *How to run a program.*)

For more information, consult *login*(1) and *getty*(8), which discuss the login sequence in more detail, and *stty*(1), which tells you how to describe the characteristics of your terminal to the system (*profile*(5) explains how to accomplish this last task automatically every time you log in).

Logging out. There are two ways to log out:

1. You can simply hang up the phone.
2. You can log out by typing an end-of-file indication (ASCII **EOT** character, usually typed as "control-d") to the shell. The shell will terminate and the **login:** message will appear again.

How to communicate through your terminal. When you type to UNIX, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a "return" (or "new-line"), as described above in *Logging in.*

UNIX terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have interspersed in it the input characters. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away *all* the saved characters.

On an input line from a terminal, the character @ "kills" all the characters typed before it. The character # erases the last character typed. Successive uses of # will erase characters back to, but not beyond, the beginning of the line; @ and # can be typed as themselves by preceding them with \ (thus, to erase a \, you need two #s). These default erase and kill characters can

be changed; see *stty*(1).

The ASCII **DC3** (control-s) character can be used to temporarily stop output. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when a **DC1** (control-q) or a second **DC3** (or any other character, for that matter) is typed. The **DC1** and **DC3** characters are not passed to any other program when used in this manner.

The ASCII **DEL** (a.k.a. "rubout") character is not passed to programs, but instead generates an *interrupt signal,* just like the "break", "interrupt", or "attention" signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed*(1), for example, catches interrupts and stops what *it* is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The *quit* signal is generated by typing the ASCII **FS** character. It not only causes a running program to terminate, but also generates a file with the "core image" of the terminated process. *Quit* is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent as to whether you have a terminal with the "new-line" function, or whether it must be simulated with a "carriage-return" and "line-feed" pair. In the latter case, all *input* "carriage-return" characters are changed to "line-feed" characters (the standard line delimiter), and a "carriage-return" and "line-feed" pair is echoed to the terminal. If you get into the wrong mode, the *stty*(1) command will rescue you.

Tab characters are used freely in UNIX source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty*(1) command will set or reset this mode. The system assumes that tabs are set every eight character positions. The *tabs*(1) command will set tab stops on your terminal, if that is possible.

How to run a program. When you have successfully logged into UNIX, a program called the shell is listening to your terminal. The shell reads the lines you type, splits them into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see *The current directory* below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space and/or tab characters.

When a program terminates, the shell will ordinarily regain control and type a **$** at you to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh*(1).

The current directory. UNIX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is by default assumed to be in that directory. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their respective owners, or by the system administrator. To change the current directory use *cd*(1).

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with /, which is the name of the *root* directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a /), until finally the file name is reached (e.g., **/usr/ae/filex** refers to file **filex** in directory **ae**, while **ae** is itself a subdirectory of **usr**; **usr** springs directly from the root directory). See *intro*(2) for a formal definition of *path name*.

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (*without* a prefixed /). Without important exception, a path name may be used anywhere a file name is required.

Important commands that modify the contents of files are *cp*(1), *mv*(1), and *rm*(1), which respectively copy, move (i.e., rename), and remove files. To find out the status of files or directories, use *ls*(1). Use *mkdir*(1) for making directories and *rmdir*(1) for destroying them.

For a fuller discussion of the file system, see the references cited at the beginning of the *INTRODUCTION* above. It may also be useful to glance through Section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Writing a program. To enter the text of a source program into a UNIX file, use *ed*(1). The four principal languages available under UNIX are C (see *cc*(1)), Fortran (see *f77*(1)), bs (a compiler/interpreter in the spirit of Basic, see *bs*(1)), and assembly language (see *as*(1)). After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file in the current directory named **a.out** (if that output is precious, use *mv*(1) to give it a less vulnerable name). If the program is written in assembly language, you will probably need to load with it library subroutines (see *ld*(1)). Fortran and C call the loader automatically; programs written in *bs*(1) are interpreted and, therefore, do not need to be loaded.

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the shell in response to the **$** prompt.

If any execution (run-time) errors occur, you will need *adb*(1) to examine the remains of your program.

Your programs can receive arguments from the command line just as system programs do; see *exec*(2).

Text processing. Almost all text is entered through the editor *ed*(1). The commands most often used to write text on a terminal are *cat*(1), *pr*(1), and *nroff*(1). The *cat*(1) command simply dumps ASCII text on the terminal, with no processing at all. The *pr*(1) command paginates the text, supplies headings, and has a facility for multi-column output. *Nroff*(1) is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file; it produces output on a typewriter-like terminal. *Troff*(1) is very similar to *nroff*(1), but produces its output on a phototypesetter (it was used to typeset this manual). There are several "macro" packages (especially the so-called *mm* package) that significantly ease the effort required to use *nroff*(1) and *troff*(1); Section 7 entries for these packages indicate where you can find their detailed descriptions.

Surprises. Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you. To communicate with another user currently logged in, *write*(1) is used; *mail*(1) will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

When you log in, a message-of-the-day may greet you before the first **$**.

CONTENTS

1. Commands and Application Programs

2. System Calls

3. Subroutines

4. Special Files

5. File Formats

6. Games

7. Miscellaneous Facilities

8. System Maintenance Procedures

PERMUTED INDEX

NAME

intro – introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, publicly-accessible commands. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.
(1C) Commands for communication with other systems.
(1G) Commands used primarily for graphics and computer-aided design.
(1M) Commands used primarily for system maintenance.

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]
where:

name	The name of an executable file.
option	– *noargleter*(*s*) or, – *argletter*<>*optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with – or, – by itself indicating the standard input.

NOTES

Plexus has made the following major changes affecting this section.

File system block size is 1024 bytes, not 512.

The system namelist is **sys3**, not **unix**.

New commands have been added. Some are based on similar commands from the University of California at Berkeley (signalled by B in the list below); others are Plexus additions (P); some are of UNIX Version 7 origin (V7). The new commands are the following:

accept(1M)	Allow or prevent LP requests. (P)
arcv6(1)	Convert archives to new format. (V7)
bbanner(1)	Another version of **banner**(1). (B)
bls(1)	List contents of directory. (B)
clear(1)	Clear terminal screen. (B)
copytape(1M)	Make an image copy of a tape. (P)
csh(1)	A shell (command interpreter) with C-like syntax. (B)
ctags(1)	Create a tags file. (B)
diction(1), explain(1)	Print wordy sentences; interactive thesaurus for *diction*. (V7)
dnld(1M)	Download program files. (P)
dumpdir(1M)	Print the names of files on a dump tape. (V7)

edit(1)	Text editor. Variant of the **ex** editor for new or casual users. (B)
enable(1)	Enable or disable LP printers. (P)
ex(1)	Text editor. (B)
head(1)	Give first few lines of a stream. (B)
icpdmp(1M)	Take a core image of the ICP and transfer to a host file. (P)
lp(1)	Sand or cancel requests to an LP line printer. (P)
lpadmin(1M)	Configure the LP spooling system. (P)
lphold(1)	Hold up print request, re-enable it. (P)
lpsched(1M)	Start or stop the LP request scheduler and move requests. (P)
lpstat(1)	Print LP status information. (P)
mkstr(1)	Create an error message file by massaging the C source. (B)
more(1)	File perusal filter for CRT viewing. (B)
node(1M)	Enable or disable foreign hosts. Available with the Network Operating System (NOS) only. (P)
openup(1M)	Keep open key directories and files. (P)
printenv(1)	Print out the environment. (B)
rmount(1M)	Mount and dismount remote file system directory. Available with the Network Operating System (NOS) only. (P)
script(1)	Make typescript of terminal session. (B)
strings(1)	Find the printable strings in an object, or other binary, file. (B)
style(1)	Analyze surface characteristics of a document. (V7)
tape(1)	Tape manipulation. (P)
topq(1M)	Put a print request at the head of the queue. (P)
tset(1)	Set terminal modes. (B)
update(1M)	Periodically update the super block. (V7)
vi(1)	Screen-oriented (visual) display editor based on **ex**. (B)
vtty(1)	Contact a remote system via the Network Operating System (NOS). (P)
xstr(1)	Extract strings from C programs to implement shared strings. (B)

Some commands, which are relevant to non-Plexus hardware only, have been deleted. These are

as(1)	PDP-11 assembler.
kun(1)	Un-assembler for KMC1/DMC11 microprocessor.
orjestat(1C)	RJE status and inquiries.
vlx(1M)	VAX console floppy interface.

In stock SYSTEM III, the **as** command is the assembler for the PDP-11; here it is the assembler for the Z8000 processor.

The command **lpd** replaces **dpd** and **odpd**.

The command **factor** is not provided because no source was provided in the stock SYSTEM III UNIX release.

SEE ALSO

getopt(1), getopt(3C).
Section 6 of this volume for computer games.
How to Get Started, at the front of this volume.

DIAGNOSTICS

Upon termination, each command returns two 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*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

Regrettably, many commands do not adhere to the aforementioned syntax.

DESCRIPTION

300, 300s - handle special functions of DASI 300 and 300s terminals

SYNOPSIS

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

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

DESCRIPTION

300 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. 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%. *300* 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*(1) 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*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

300 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.

300(1)

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.

The *neqn*(1) names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(7).

SEE ALSO

450(1), eqn(1), graph(1G), mesg(1), stty(1), tabs(1), tbl(1), tplot(1G), troff(1), greek(7).

BUGS

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.

NAME

4014 - paginator for the Tektronix 4014 terminal

SYNOPSIS

4014 [**-t**] [**-n**] [**-c**N] [**-p**L] [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 Don't 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(1), tc(1), troff(1).

450(1)

NAME

450 - handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

450 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*(1). *450* can be used 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.

450 can be used with the *nroff*(1) **-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 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.

The *neqn*(1) names of, and resulting output for, the Greek and special characters supported by *450* are shown in *greek*(7).

SEE ALSO

300(1), eqn(1), graph(1G), mesg(1), stty(1), tabs(1), tbl(1), tplot(1G), troff(1), greek(7).

BUGS

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.

NAME

accept, reject – allow/prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [**–r**[reason]] destinations

DESCRIPTION

Accept allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

Reject prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject*.

–r[*reason*] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next **–r** option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the **–r** option is not present or the **–r** option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).

NAME

acct - overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usrlib/acct/acctdusg [**-u** file] [**-p** file] > **dtmp-file**

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp [name[line]] >>**/usr/adm/wtmp**

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. *Acctsh*(1M) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into **/usr/adm/utmp**, as described in *utmp*(5). The programs described in *acctcon*(1M) convert this file into session and charging records, which are then summarized by *acctmerg*(1M).

Process accounting is performed by the UNIX kernel. Upon termination of a process, one record per process is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data may be examined using *acctcom*(1).

Process accounting and connect time accounting (or any accounting records in the format described in *acct*(5)) can be merged and summarized into total accounting records by *acctmerg* (see **tacct** format in *acct*(5)). *Prtacct* (see *acctsh*(1M)) is used to format any or all accounting records.

Acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

Acctdusg reads its standard input (usually from **find / -print**) and computes disk resource consumption (including indirect blocks) by login. If **-u** is given, records consisting of those file names for which *acctdusg* charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If **-p** is given, *file* is the name of the password file. This option is not needed if the password file is **/etc/passwd**.

Accton alone turns process accounting off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records (see *acct*(2) and *acct*(5)).

Acctwtmp writes a *wtmp*(5) record to its standard output. The record contains the current time, *name*, and *line*. If *line* is omitted, a value is emitted that is interpreted by other programs as a reboot. For more precise accounting, the following are recommended for use in reboot and shutdown procedures, respectively:

```
acctwtmp `uname` >>/usr/adm/wtmp
acctwtmp reason >>/usr/adm/wtmp
```

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/pacct	current process accounting file
/usr/adm/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).
The UNIX Accounting System by H. S. McCreary.

DIAGNOSTICS

"permission denied - you must be root or adm" may be printed by *accton* if adm does not own **/usr/adm/pacct**.

NAME

acctcms - command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [options] files

DESCRIPTION

Acctcms reads one or more *files*, normally in the form described in *acct*(5). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The *options* are:

-a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, and "hog factor", as in *acctcom*(1). Output is normally sorted by total kcore-minutes.

-c Sort by total CPU time, rather than total kcore-minutes.

-j Combine all commands invoked only once under "***other".

-n Sort by number of command invocations.

-s Any file names encountered hereafter are already in internal summary format.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file ... >today
cp total previoustotal
acctcms -s today previoustotal >total
acctcms -a -s today
```

SEE ALSO

acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

acctcom [[options][file]] . . .

DESCRIPTION

Acctcom reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(5) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: **1** for *fork* without *exec*) and **STAT** (the system exit status).

The command name is prepended with a **#** if it was executed with *super-user* privileges. If a process is not associated with a known terminal, a **?** is printed in the **TTYNAME** field.

If no *files* are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using **&** in the shell), **/usr/adm/pacct** is read, otherwise the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several files, in which case all but the current will be found in **/usr/adm/pacct?**. The *options* are:

-b Read backwards, showing latest commands first.

-f Print the *fork/exec* flag and system exit status columns in the output.

-h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:
(total CPU time)/(elapsed time).

-i Print columns containing the I/O counts in the output.

-k Instead of memory size, show total kcore-minutes.

-m Show mean core size (the default).

-r Show CPU factor (user time/(system-time + user-time).

-t Show separate system and user CPU times.

-v Exclude column headings from the output.

-l *line* Show only processes belonging to terminal **/dev/***line*.

-u *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a **#** which designates only those processes executed with *super-user* privileges, or **?** which designates only those processes associated with unknown user IDs.

-g *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

-d *mm/dd* Any *time* arguments following this flag are assumed to occur on the given month and day, rather than during the last 24 hours. This is needed for looking at old files.

-s *time* Show only those processes that existed on or after *time*, given in the form *hr*:*min*:*sec*. The :*sec* or :*min*:*sec* may be omitted.

-e *time* Show only those processes that existed on or before *time*. Using the same *time* for both **-s** and **-e** shows the processes that existed at *time*.

-n *pattern* Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that + means one or more occurrences.

-H *factor* Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **-h** above.

-O *time* Show only those processes with operating system CPU time that exceeds *time*.

-C *time* Show only those processes that exceed *time* that indicates the total CPU time.

ACCTCOM(1)

Listing options together has the effect of a logical *and*.

FILES

/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), ps(1), runacct(1M), su(1), acct(2), acct(5), utmp(5).

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes.

NAME

acctcon - connect-time accounting

SYNOPSIS

acctcon1 [options]

acctcon2

DESCRIPTION

Acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/usr/adm/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The *options* are:

-p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

-t *Acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **-t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

-l *file* *File* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Both hang-up and termination of the login shell generate a logoff record, so that the number of logoffs is often twice the number of sessions.

-o *file* *File* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

Acctcon1 reads the file **/usr/lib/acct/holidays** for the current year and the current year's holidays. It assumes default values if the file does not exist. See *holidays*(5).

Acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see **tacct** format in *acct*(5)).

EXAMPLES

These commands are typically used as shown below. The file **ctmp** is created only for the use of *acctprc*(1M) commands:

```
acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

/usr/adm/wtmp
/usr/lib/acct/holidays

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), holidays(5), utmp(5).

BUGS

The line usage report is confused by date changes. Use *wtmpfix* (see *fwtmp*(1M)) to correct this situation.

NAME

acctmerg - merge or add total accounting files

SYNOPSIS

acctmerg [options] [file] . . .

DESCRIPTION

Acctmerg reads its standard input and up to nine additional files, all in the **tacct** format (see *acct*(5)), or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys. *Options* are:

- **-a** Produce output in ASCII version of **tacct**.
- **-i** Input files are in ASCII version of **tacct**.
- **-p** Print input with no processing.
- **-t** Produce a single record that totals all input.
- **-u** Summarize by user ID, rather than user ID and name.
- **-v** Produce output in verbose ASCII format, with more precise notation for floating point numbers.

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg -v <file1 >file2
        edit file2 as desired ...
acctmerg -a <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

NAME

acctprc - process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

DESCRIPTION

Acctprc1 reads input in the form described by *acct*(5), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in 2048-byte units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

Acctprc1 reads the file **/usr/lib/acct/holidays** for the current year and the current year's holidays. It assumes default values if the file does not exist. See *holidays*(5).

Acctprc2 reads records in the form written by *acctprc1*, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records. It writes in the form of **tacct**. See *acct*(5).

These commands are typically used as shown below:

```
acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct
```

FILES

/etc/passwd

NOTES

Plexus uses 2048-byte units for mean memory size.

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), holidays(5), utmp(5).

BUGS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from *cron*(1M), for example. More precise conversion can be done by faking login sessions on the console via the *acctwtmp* program in *acct*(1M).

NAME

acctsh - shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee login-name number

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily

/usr/lib/acct/prtacct file ["heading"]

/usr/lib/acct/runacct [mmdd] [mmdd state]

/usr/lib/acct/shutacct ["reason"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct [**on** | **off** | **switch**]

DESCRIPTION

Chargefee is invoked to charge *number* dollars to *login-name*. A record is written to **/usr/adm/fee**, to be merged with other accounting records during the night.

Ckpacct is initiated via *cron*. It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* will be invoked with argument *switch*.

Dodisk is invoked by *cron* to perform the disk accounting functions.

Lastlogin is invoked by *runacct* to update **/usr/adm/acct/sum/loginlog**, which shows the last date on which each person logged in.

Monacct should be invoked once each month or each accounting period. *Number* indicates which month or period it is. It creates summary files in **/usr/adm/acct/fiscal** and restarts summary file in **/usr/adm/acct/sum**. *Nulladm* creates *file* with mode 644 and insures owner is **adm**. It is called by *lastlogin*, *runacct*, and *turnacct*.

Prctmp can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon1* (see *acctcon*(1M)).

Prdaily is invoked by *runacct* to print a report of the previous day's accounting. The report resides in **/usr/adm/acct/sum/rprt***xxxx* where *xxxx* is the month and day of the report. The daily accounting reports may be printed (by the command "cat /usr/adm/acct/sum/rprt*") as often as desired and they must be explicitly deleted when no longer needed.

Prtacct can be used to format and print any total accounting file.

Runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see *runacct*(1M).

Shutacct should be invoked during a system shutdown to turn process accounting off and append a "reason" record to **/usr/adm/wtmp**. *Startup* should be called by *rc*(8) to turn the accounting on whenever the system is brought up.

Turnacct is an interface to *accton* (see *acct*(1M)) to turn process accounting **on** or **off**. The **switch** argument moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct[1-9]**, turns accounting off, then turns it back on again. This procedure is called by *ckpacct* via the *cron* to keep the **pacct** file size smaller.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct[1-9]	used if pacct gets large and during execution of daily accounting procedure
/usr/adm/wtmp	login/logoff summary
/usr/adm/wtmp[1-9]	used during daily accounting procedure
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/acct/sum	summary directory, should be saved

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

NAME

adb - debugger

SYNOPSIS

adb [**-w**] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is **a.out**. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core**.

Requests to *adb* are read from the standard input and responses are to the standard output. If the **-w** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

[*address*] [, *count*] [*command*] [;]

If *address* is present then *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see *ADDRESSES*.

EXPRESSIONS

. The value of *dot*.

\+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

integer An octal number if *integer* begins with a 0; a hexadecimal number if preceded by #; otherwise a decimal number.

integer.fraction
A 32 bit floating point number.

'cccc ' The ASCII value of up to 4 characters. \ may be used to escape a '.

< name The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (see *VARIABLES*) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil*. The register names are **r0 ... r15 fcw seg pc** for the Z8000, **d0 ... d7 a0 ... a7 ps pc** for the MC68000.

symbol A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ or ~ will be prepended to *symbol* if needed.

_ symbol In C, the "true name" of an external symbol begins with _. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

routine.name
The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently

activated C stack frame corresponding to *routine*. (Not implemented in the MC68000.)

(*exp*) The value of the expression *exp*.

Monadic operators:

**exp* The contents of the location addressed by *exp* in *corfil*.

@exp The contents of the location addressed by *exp* in *objfil*.

-exp Integer negation.

~exp Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1+e2 Integer addition.

e1-e2 Integer subtraction.

*e1*e2* Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

e1#e2 *E1* rounded up to the next multiple of *e2*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands **?** and / may be followed by *; see *ADDRESSES* for further details.)

?*f* Locations starting at *address* in *objfil* are printed according to the format *f*. *dot* is incremented by the sum of the increments for each format letter (q.v.).

/*f* Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for **?**.

=*f* The value of *address* itself is printed in the styles indicated by the format *f*. (For i format **?** is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows:

o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O 4 Print 4 bytes in octal.
q 2 Print in signed octal.
Q 4 Print long signed octal.
d 2 Print in decimal.
D 4 Print long decimal.
x 2 Print 2 bytes in hexadecimal.
X 4 Print 4 bytes in hexadecimal.
u 2 Print as an unsigned decimal number.
U 4 Print long unsigned decimal.
f 4 Print the 32 bit value as a floating point number.
F 8 Print double floating point.
b 1 Print the addressed byte in octal.
c 1 Print the addressed character.

ADB(1)

C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.

s *n* Print the addressed characters until a zero character is reached.

S *n* Print a string using the @ escape convention. *n* is the length of the string including its zero terminator.

Y 4 Print 4 bytes in date format (see *ctime*(3C)).

i n Print as Z8000 instructions. *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.

Z n Prints as Z8000 assembler listing. (Not available on the MC68000.)

a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/ local or global data symbol
? local or global text symbol
= local or global absolute symbol

p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.

t 0 When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

r 0 Print a space.

n 0 Print a new-line.

"...**"** 0 Print the enclosed string.

^ *Dot* is decremented by the current increment. Nothing is printed.

\+ *Dot* is incremented by 1. Nothing is printed.

\- *Dot* is decremented by 1. Nothing is printed.

new-line

Repeat the previous command with a *count* of 1.

[**?**/]**l** *value mask*

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted then -1 is used.

[**?**/]**w** *value ...*

Write the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[**?**/]**m** *b1 e1 f1*[**?**/]

New values for (*b1, e1, f1*) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the **?** or / is followed by * then the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If the list is terminated by **?** or / then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, **/m?** will cause / to refer to *objfil*.)

>*name*

Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following **!**.

$*modifier*

Miscellaneous commands. The available *modifiers* are:

<*f* Read commands from the file *f* and return.

>*f* Send output to the file *f*, which is created if it does not exist.

r Print the general registers and the instruction addressed by **pc**. *Dot* is set to **pc**.

f Print the registers as double precision values.

b Print all breakpoints and their associated counts and commands.

c C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **r14** (Z8000) or **fp** (MC68000). If **C** is used then the names and (16 bit) values of all automatic and static variables are printed for each active function (Z8000 only). If *count* is given then only the first *count* frames are printed.

e The names and values of external variables are printed.

w Set the page width for output to *address* (default 80).

s Set the limit for symbol matches to *address* (default 255).

o All integers input are regarded as octal.

x Changes default output to hexadecimal.

d Reset integer input as described in *EXPRESSIONS*.

q Exit from *adb*.

v Print all non zero variables in octal.

m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

bc Set breakpoint at *address*. The breakpoint is executed *count*-1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero then the breakpoint causes a stop.

d Delete breakpoint at *address*.

x Delete all breakpoints.

e If *address* is given, set breakpoint upon exit from the routine on stack whose address is *address*. If *address* is not given, set breakpoint upon exit from current routine. This is useful for looking at the values returned (in **r7** or **rr6** or **rg4**) by a procedure.

p Similar to **b**, except that *address* is assumed to be a procedure name, and the breakpoint is positioned after the stack frame has been set up.

q Does for **p** what **d** does for **b**.

r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.

cs The subprocess is continued with signal *s* (see *signal*(2)). If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.

ss As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

0 The last value printed.
1 The last offset part of an instruction source.
2 The previous value of variable 1.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a **core** file then these values are set from *objfil*.

b The base address of the data segment.
d The data segment size.
e The entry point.
m The "magic" number (0405, 0407, 0410 or 0411).
s The stack segment size.
t The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows:

$b1 \leq address < e1 \Rightarrow file\ address = address + f1 - b1$

otherwise

$b2 \leq address < e2 \Rightarrow file\ address = address + f2 - b2,$

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a **?** or **/** is followed by an ***** then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

In order for *adb* to be used on large files all appropriate values are kept as signed 32 bit integers.

FILES

/dev/mem
/dev/swap
a.out
core

NOTES

Plexus *adb* differs from the standard SYSTEM III *adb* in the following ways.
The number of registers and certain register names have changed.
Format option **i** prints Z8000 or MC68000 instructions instead of PDP-11 instructions.
A new **z** format option is provided for disassembly (Z8000 only).
The **a** (ALGOL stack backtrace) modifier is not supported.
A new modifier **x** changes default output to hexadecimal.
Four new modifier options are provided: **e**, **p**, **q**, and **x** (**e** available on Z8000 only).

SEE ALSO

ptrace(2), a.out(5), core(5).

DIAGNOSTICS

"Adb" when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or

returned nonzero status.

BUGS

A breakpoint set at the entry point is not effective on initial entry to the program.
Local variables whose names are the same as an external variable may foul up the accessing of the external.
The MC68000 disassembly (**i** option) is not always correct.

NAME

admin - create and administer SCCS files

SYNOPSIS

admin [**-n**] [**-i**[name]] [**-r**rel] [**-t**[name]] [**-f**flag[flag-val]] [**-d**flag[flag-val]] [**-a**login] [**-e**login] [**-m**[mrlist]] [**-y**[comment]] [**-h**] [**-z**] files

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments, which begin with **-**, and named files (note that SCCS file names must begin with the characters **s.**). If a named file doesn't exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

-n This keyletter indicates that a new SCCS file is to be created.

-i[*name*] The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **-r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files require that they be created empty (no **-i** keyletter). Note that the **-i** keyletter implies the **-n** keyletter.

-r*rel* The *rel*ease into which the initial delta is inserted. This keyletter may be used only if the **-i** keyletter is also used. If the **-r** keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

-t[*name*] The *name* of a file from which descriptive text for the SCCS file is to be taken. If the **-t** keyletter is used and *admin* is creating a new SCCS file (the **-n** and/or **-i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a **-t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **-t** keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

-f*flag* This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:

b Allows use of the **-b** keyletter on a *get*(1) command to create branch deltas.

c*ceil* The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified **c** flag is 9999.

f*floor* The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified **f** flag is 1.

d*SID* The default delta number (SID) to be used by a *get*(1) command.

i Causes the "No id keywords (ge6)" message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file.

j Allows concurrent *get*(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* A *list* of releases to which deltas can no longer be made (**get -e** against one of these "locked" releases fails). The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= *RELEASE NUMBER* | **a**

The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.

n Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file preventing branch deltas from being created from them in the future.

q*text* User definable text substituted for all occurrences of the keyword in SCCS file text retrieved by *get*(1).

m*mod* *Mod*ule name of the SCCS file substituted for all occurrences of the admin.1 keyword in SCCS file text retrieved by *get*(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

t*type* *Type* of module in the SCCS file substituted for all occurrences of keyword in SCCS file text retrieved by *get*(1).

v[*pgm*]
Causes *delta*(1) to prompt for Modification Request (*MR*) numbers as the reason for creating a delta. The optional value specifies the name of an *MR* number validity checking program (see *delta*(1)). (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null).

-d*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The **-d** keyletter may be specified only when processing existing SCCS files. Several **-d** keyletters may be supplied on a single *admin* command. See the **-f** keyletter for allowable *flag* names.

-l*list* A *list* of releases to be "unlocked". See the **-f** keyletter for a description of the **l** flag and the syntax of a *list*.

-a*login* A *login* name, or numerical UNIX group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters may be used on a single *admin* command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.

-e*login* A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **e** keyletters may be used on a single *admin* command line.

-y[*comment*] The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the **-y** keyletter results in a default comment line being inserted in the form:

date and time created *YY/MM/DD HH:MM:SS* by *login*

The **-y** keyletter is valid only if the **-i** and/or **-n** keyletters are specified (i.e., a new SCCS file is being created).

-m[*mrlist*] The list of Modification Requests (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The **v** flag must be set and the *MR* numbers are validated if the **v** flag has a value (the name of an *MR* number validation program). Diagnostics will occur if the **v** flag is not set or *MR* validation fails.

-h Causes *admin* to check the structure of the SCCS file (see *sccsfile*(5)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

FILES

The last component of all SCCS file names must be of the form **s.***file-name*. New SCCS files are given mode 444 (see *chmod*(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called **x.***file-name*, (see *get*(1)), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed*(1). *Care must be taken!* The edited file should *always* be processed by an **admin -h** to check for corruption followed by an **admin -z** to generate a proper check-sum. Another **admin -h** is recommended to ensure the SCCS file is valid.

Admin also makes use of a transient lock file (called **z.***file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get*(1) for further information.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

AR(1)

NAME

ar - archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose.

Ar can read archive files produced in either Z8000 or 68000 format (see *ar*(5)). However, when *ar* creates an archive, it always creates the header in the format of the local system. A conversion program exists to convert Z8000 archives to 68000 archive format (see *arcv*(1)). This feature is useful only for source archive files. Individual files are inserted without conversion into the archive file.

Key is one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcl**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

v Verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.

c Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally *ar* places its temporary files in the directory /**tmp**. This option causes them to be placed in the local directory.

FILES

/tmp/v* temporaries

NOTES

Plexus *ar* reads archive files produced in Z8000 or 68000 format (not PDP-11 or VAX format).

SEE ALSO

arcv(1), ld(1), lorder(1), ar(5).

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

arcv6 - convert archives to new format

SYNOPSIS

arcv6 file ...

DESCRIPTION

Arcv6 converts archive files (see *ar*(1), *ar*(5)) from 6th edition to 7th edition format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177555 at the start; new archives have 0177545.

FILES

/tmp/v*, temporary copy

SEE ALSO

ar(1), ar(5).

NAME

as - Z8000 assembler

SYNOPSIS

as [-l] [-m] [-n] [-o OBJFILE] [-u] [-v] [-x] FILE

DESCRIPTION

As assembles the concatenation of the named files. The options may be placed in any order.

The "-l" option causes a listing to be created on the standard output.

The "-m" option places a tab at the left margin of each line when the listing option "-l" has been selected.

The "-n" option causes a listing to be created on the standard output, but causes a narrower listing than the "-l" option. This is sometimes useful for 80 column paper, but does not contain line numbers.

The "-u" option causes all undefined symbols in the current assembly to be made undefined-external.

The "-v" option puts the assembler in verbose mode, which causes the files to be listed during the first pass, along with the errors encountered.

The "-x" option causes a cross refererence to be created on the file named "xref". The cross reference consists of an alphabetical listing of all user symbols along with the value, line number, and line numbers for all references.

The output of the assembler is by default placed on the file "a.out" in the current directory; the "-o" flag causes the output to be placed on the named file. If there were no unresolved external references, and no errors detected, the output file is marked executable; otherwise, if it is produced at all, it is made non-executable.

FILES

/usr/tmp/as*	temporary
a.out	object
xref	cross reference

SEE ALSO

ld(1), nm(1), adb(1), a.out(5)
UNIX Z8000 Assembler Reference Manual by Craig C. Forney

DIAGNOSTICS

Diagnostics are meant to be self explanatory and are accompanied by either the offending file or the appropriate line number. If the listing option is used, then the error messages also are placed in the listing file.

BUGS

Cross reference ("-x" option) is not implemented. Only a single file may be assembled at a time.

NAME

as - MC68000 assembler

SYNOPSIS

as [-g] [-o OBJFILE] [-e] [-l] FILE

DESCRIPTION

As assembles the concatenation of the named files. The options may be placed in any order.

The "-l" option causes a listing to be created in a file whose name has ".list" substituted for ".s".

The "-g" option causes undefined symbols to be made global.

The "-e" option causes only external symbols to be placed in the ".o" file.

The output of the assembler is by default placed on the file with ".o" substituted for ".s" in the current directory; the "-o" flag causes the output to be placed on the named file. If there were no unresolved external references, and no errors detected, the output file is marked executable; otherwise, if it is produced at all, it is made non-executable.

FILES

/usr/tmp/as*	temporary
a.out	object
xref	cross reference

SEE ALSO

ld(1), nm(1), adb(1), a.out(5)
UNIX MC68000 Assembler Reference Manual

DIAGNOSTICS

Diagnostics are meant to be self explanatory and are accompanied by either the offending file or the appropriate line number. If the listing option is used, then the error messages also are placed in the listing file.

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [**-Fc**] [**-f** file] [prog] [files]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as **-f** *file*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

Files are read in order; if there are no files, the standard input is read. The file name **-** means the standard input. Each 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 made up of fields separated by white space. (This default can be changed by using FS, see below). The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the form:

pattern { action }

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole 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. 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 *>expr* is present), 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*(3S)).

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf*(*fmt*, *expr*, *expr*, ...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see *grep*(1)). 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 the next occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

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

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

```
BEGIN { FS = c }
```

or by using the **-F***c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default **%.6g**).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up 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 }
```

SEE ALSO

grep(1), lex(1), sed(1).
Awk-A Pattern Scanning and Processing Language by A. V. Aho, B. W. Kernighan, and P. J. Weinberger.

BUGS

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.

NAME

banner - make posters

SYNOPSIS

banner strings

DESCRIPTION

Banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

bbanner(1).

NAME

basename, dirname - deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

Basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (` `) within shell procedures.

Dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, 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`
```

The following example will set the shell variable **NAME** to **/usr/src/cmd**:

```
NAME=`dirname /usr/src/cmd/cat.c`
```

SEE ALSO

sh(1).

NAME

bbanner - print large banner on printer

SYNOPSIS

/usr/plx/bbanner [**-w***n*] message ...

DESCRIPTION

Bbanner prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If **-w** is given, the output is scrunched down from a width of 132 to *n* , suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a DECwriter or other 300 baud terminal will do.

NOTES

The Plexus version of this command is based on the one developed at the University of California at Berkeley.

SEE ALSO

banner(1).

BUGS

Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The **-w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

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 letter a-z, E means expression, S means statement.

Comments

are enclosed in /* and */.

Names

simple variables: L
array elements: L [E]
The words "ibase", "obase", and "scale"

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)

Operators

+ - * / % ^ (% is remainder; ^ is power)
++ -- (prefix and postfix; apply to names)
== <= >= != < >
= =+ =- =* =/ =% =^

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

BC(1)

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 new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(1). 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*(1), which it invokes automatically, unless the **-c** (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

EXAMPLE

```
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
        }
}
```

defines a function to compute an approximate value of the exponential function and

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

prints approximate values of the exponential function of the first ten integers.

FILES

/usr/lib/lib.b	mathematical library
/usr/bin/dc	desk calculator proper

SEE ALSO

dc(1).
BC - An Arbitrary Precision Desk-Calculator Language
by L. L. Cherry and R. Morris.

BUGS

No **&&**, **||** yet.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

NAME

bcopy - interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

Bcopy dates from a time when neither the UNIX file system nor the DEC disk drives were as reliable as they are now. *Bcopy* copies from and to files starting at arbitrary block (512-byte) boundaries.

The following questions are asked:

to: (you name the file or device to be copied to).
offset: (you provide the starting "to" block number).
from: (you name the file or device to be copied from).
offset: (you provide the starting "from" block number).
count: (you reply with the number of blocks to be copied).

After **count** is exhausted, the **from** question is repeated (giving you a chance to concatenate blocks at the **to**+**offset**+**count** location). If you answer **from** with a carriage return, everything starts over.

Two consecutive carriage returns terminate *bcopy*.

SEE ALSO

cpio(1), dd(1).

NAME

bdiff - big diff

SYNOPSIS

bdiff file1 file2 [n] [**-s**]

DESCRIPTION

Bdiff is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is **-**, the standard input is read. The optional **-s** (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*. If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

bfs - big file scanner

SYNOPSIS

bfs [-] name

DESCRIPTION

Bfs is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) 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*(1) 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 - suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides / and **?**: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regex*(3X)). There is 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**, **n**, **p**, **q**, **w**, =, **!** and null commands operate as described under *ed*. 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* file name. 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*, which, if necessary, is created mode 666. 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 new-line, 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, **.** 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 someplace 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 at most *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
```

will all print the first 100 lines.

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

would globally search for the characters **100** and print each line containing a match. To escape the special meaning of **%**, a **** must precede it.

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

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.
Another feature of the **xv** command is that the first line of output from a UNIX command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example:

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

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a ****.

```
xv7\!date
```

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

xbz *label*

xbn *label*
These two commands will test the last saved *return code* from the execution of a UNIX command (*!command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string **size**.

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

xc [*switch*]
If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it isn't. 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(1), ed(1), regex(3X).

DIAGNOSTICS

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

BUGS

Bfs does unpredictable things with lines greater than 255 characters.

NAME

bls - list contents of directory

SYNOPSIS

/usr/plx/bls [**-abcdfgilmqrstux1CFR**] name ...

DESCRIPTION

For each directory argument, *bls* lists the contents of the directory; for each file argument, *bls* 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, but file arguments appear before directories and their contents.

There are three major listing formats. The format chosen depends on whether the output is going to a teletype, and may also be controlled by option flags. The default format for a teletype is to list the contents of directories in multi-column format, with the entries sorted down the columns. (Files that are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. This is because the individual file names may be arbitrarily long.) If the standard output is not a teletype, the default format is to list one entry per line. Finally, there is a stream output format in which files are listed across the page, separated by ',' characters. The **-m** flag enables this format.

This command has many options:

-l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers.

-t Sort by time modified (latest first) instead of by name, as is normal.

-a List all entries; usually '.' and '..' are suppressed.

-s Give size in (1024-byte) blocks, including indirect blocks, for each entry.

-d If argument is a directory, list only its name, not its contents (mostly used with **-l** to get status on directory).

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-u Use time of last access instead of last modification for sorting (**-t**) or printing (**-l**).

-c Use time of file creation for sorting or printing.

-i Print i-number in first column of the report for each file listed.

-f Force each argument to be interpreted as a directory and list 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 Give group ID instead of owner ID in long listing.

-m Force stream output format.

-1 Force one entry per line output format, e.g. to a teletype.

-C Force multi-column output, e.g. to a file or a pipe.

-q Force printing of non-graphic characters in file names as the character '?'; this normally happens only if the output device is a teletype.

-b Force printing of non-graphic characters to be in the \ddd rotation, in octal.

-x Force columnar printing to be sorted across rather than down the page; this is the default if the last character of the name the program is invoked with is an 'x'.

-F Cause directories to be marked with a trailing '/' and executable files to be marked with a trailing '*'; this is the default if the last character of the name the program is invoked with is a 'f'.

-R Recursively list subdirectories encountered.

The mode printed under the **-l** option contains 11 characters, which are interpreted as follows: the first character is

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;
m if the entry is a multiplexor-type character special file;
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. 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*(1) 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 printed.

FILES

/etc/passwd	to get user ID's for 'bls -l'.
/etc/group	to get group ID's for 'bls -g'.

NOTES

This command is based on the *ls* command from the University of California at Berkeley.

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable, because "bls -s" is very different from "bls -s | lpr". On the other hand, not doing this setting would make old shell scripts that used *ls* almost certain not to work.

Column widths choices are poor for terminals that can tab.

NAME

bs - a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [file [args]]

DESCRIPTION

Bs is a remote descendant of Basic and Snobol4 with a little C language thrown in. *Bs* is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the *trace* and *dump* statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see *compile* and *execute* below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

Bs programs are made up of input lines. If the last character on a line is a \, the line is continued. *Bs* accepts lines of the following form:

```
statement
label  statement
```

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A label in execute mode (see below) produces an "invalid expression" message.

A *bs* statement is either an expression or a keyword followed by zero or more expressions. Some keywords (*clear*, *compile*, *!*, *execute*, *include*, *ibase*, *obase*, and *run*) are always executed as they are compiled.

Statement Syntax:

expression
: The expression is executed for its side effects (value, assignment or function call). The details of expressions follow the description of statement types below.

break
: *Break* exits from the inner-most *for*/*while* loop.

clear
: Clears the symbol table and compiled statements. *Clear* is executed immediately.

compile [expression]
: Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A *clear* is associated with this latter case. *Compile* is executed immediately.

continue
: *Continue* transfers to the loop-continuation of the current *for*/*while* loop.

dump
: The name and current value of every non-local variable is printed. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.

exit [expression]
: Return to system level. The expression is returned as process status.

execute
: Change to immediate execution mode (an interrupt has a similar effect). This statement

does not cause stored statements to execute (see *run* below).

for name = expression expression statement
for name = expression expression
...
next

for expression , expression , expression statement
for expression , expression , expression
...
next

The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

fun f([a, ...]) [v, ...]
...
nuf

Fun defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn

A way to signal the failure of a user-written function. See the interrogation operator (**?**) below. If interrogation is not present, *freturn* merely returns zero. When interrogation *is* active, *freturn* transfers to that expression (possibly by-passing intermediate function returns).

ibase *N*

Ibase sets the input base (radix) to *N*. The only supported values for *N* are **8**, **10** (the default), and **16**. Hexadecimal values 10-15 are entered as **a-f**. A leading digit is required (i.e., **f0a** must be entered as **0f0a**). *Ibase* (and *obase*, below) are executed immediately.

goto name

Control is passed to the internally stored statement with the matching label.

if expression statement
if expression
...
[**else**
...]
fi

The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings **0** and **""** (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The elision of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if* ... *elif* ... [*else* ...] sequence.

include expression

The expression must evaluate to a file name. The file must contain *bs* Such statements become part of the program being compiled. source statements. *Include* statements may not be nested.

obase *N*

Obase sets the input base to *N* (see *ibase* above).

onintr label
onintr
The *onintr* command provides program control of interrupts. In the first form, control will pass to the label given, just as if a *goto* had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause *bs* to terminate.

return [expression]
The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run
The random number generator is reset. Control is passed to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop
Execution of internal statements is stopped. *Bs* reverts to immediate mode.

trace [expression]
The *trace* statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace* expression value.

while expression statement
while expression
...
next
While is similar to *for* except that only the conditional expression for loop-continuation is given.

! shell command
An immediate escape to the Shell.

...
This statement is ignored. It is used to interject commentary in a program.

Expression Syntax:

name
A name is used to specify a variable. Names are composed of a letter (upper or lower case) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open*( ) below).

name **(** [expression [, expression] ...] **)**
Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value.

name **[** expression [, expression] ... **]**
This syntax is used reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; **a[1,2]** is the same as **a[1][2]**. The truncated expressions are restricted to values between 0 and 32767.

number
A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an **e** followed by a possibly signed exponent.

string
Character strings are delimited by " characters. The \ escape character allows the double quote (\"), new-line (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. Otherwise, \ stands for itself.

(expression)
Parentheses are used to alter the normal order of evaluation.

(expression, expression [, expression ...]) [expression]
The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

(False, True)[a == b]

has the value **True** if the comparison is true.

? expression
The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the *Programming Tips* section below), the result of the *eval* built-in function, and for checking the return from user-written functions (see *freturn*). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

- expression
The result is the negation of the expression.

++ name
Increments the value of the variable (or array reference). The result is the new value.

-- name
Decrements the value of the variable. The result is the new value.

! expression
The logical negation of the expression. Watch out for the shell escape command.

expression *operator* expression
Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence):

=
= is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

_
_ (underscore) is the concatenation operator.

& |
& (logical and) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.

< <= > >= == !=
The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as

follows: $a>b>c$ is the same as $a>b$ & $b>c$. A string comparison is made if both operands are strings.

+ -
Add and subtract.

*** / %**
Multiply, divide, and remainder.

^
Exponentiation.

Built-in Functions:

Dealing with arguments

arg(i)
is the value of the *i*-th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument (*arg*(0) returns **bs**).

narg( )
returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

abs(x)
is the absolute value of *x*.

atan(x)
is the arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.

ceil(x)
returns the smallest integer not less than *x*.

cos(x)
is the cosine of *x* (radians).

exp(x)
is the exponential function of *x*.

floor(x)
returns the largest integer not greater than *x*.

log(x)
is the natural logarithm of *x*.

rand( )
is a uniformly distributed random number between zero and one.

sin(x)
is the sine of *x* (radians).

sqrt(x)
is the square root of *x*.

String operations

size(s)
the size (length in bytes) of *s* is returned.

format(f, a)
returns the formatted value of *a*. *F* is assumed to be a format specification in the style of *printf*(3S). Only the **%...f**, **%...e**, and **%...s** types are safe.

index(x, y)
returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

trans(s, f, t)
Translates characters of the source *s* from matching characters in *f* to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

substr(s, start, width)
returns the sub-string of *s* defined by the *start*ing position and *width*.

match(string, pattern)
mstring(n)
The *pattern* is similar to the regular expression syntax of the *ed*(1) command. The characters **.**, **[**, **]**, **^** (inside brackets), ***** and **$** are special. The *mstring* function returns the *n*-th (1 <= *n* <= 10) substring of the subject that occurred between pairs of the pattern symbols **\(** and **\)** for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with **^**). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a-z]\)") == 6
mstring(1) == "b"
```

File handling

open(name, file, function)
close(name)
The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may be **1)** a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively, **2)** a string representing a file name, or **3)** a string beginning with an **!** representing a command to be executed (via *sh -c*). The *function* argument must be either **r** (read), **w** (write), **W** (write without new-line), or **a** (append). After a *close*, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

access(s, m)
executes *access*(2).

ftype(s)
returns a single character file type indication: **f** for regular file, **d** for directory, **b** for block special, or **c** for character special.

Tables

table(name, size)

A table in *bs* is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a *bs* variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. *Bs* prints an error message and stops on table overflow.

item(name, i)

key()

The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the "subscript" of the previous *item* call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table, for example:

```
table("t", 100)
 ...
# If word contains "party", the following expression adds one to the count
# of that word:
++t[word]
 ...
# To print out the the key/value pairs:
for i = 0, ?(s = item(t, i)), ++i  if key()  put = key()_":"_s
```

iskey(name, word)

The *iskey* function tests whether the key **word** exists in the table **name** and returns one for true, zero for false.

Odds and ends

eval(s)

The string argument is evaluated as a *bs* expression. The function is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++"_ name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control *bs* error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a *goto* to the label *L* (if it exists):

```
label="L"
if !(?eval("goto "_ label)) puterr = "no label"
```

plot(request, args)

The *plot* function produces output on devices recognized by *tplot*(1G). The *requests* are as follows:

Call	*Function*
plot(0, term)	causes further *plot* output to be piped into *tplot*(1G) with an argument of -T*term*.
plot(1)	"erases" the plotter.
plot(2, string)	labels the current point with *string*.

plot(3, x1, y1, x2, y2)	draws the line between (*x1*,*y1*) and (*x2*,*y2*).
plot(4, x, y, r)	draws a circle with center (*x*,*y*) and radius *r*.
plot(5, x1, y1, x2, y2, x3, y3)	draws an arc (counterclockwise) with center (*x1*,*y1*) and endpoints (*x2*,*y2*) and (*x3*,*y3*).
plot(6)	is not implemented.
plot(7, x, y)	makes the current point (*x*,*y*).
plot(8, x, y)	draws a line from the current point to (*x*,*y*).
plot(9, x, y)	draws a point at (*x*,*y*).
plot(10, string)	sets the line mode to *string*.
plot(11, x1, y1, x2, y2)	makes (*x1*,*y1*) the lower left corner of the plotting area and (*x2*,*y2*) the upper right corner of the plotting area.
plot(12, x1, y1, x2, y2)	causes subsequent x (y) coordinates to be multiplied by *x1* (*y1*) and then added to *x2* (*y2*) before they are plotted. The initial scaling is **plot(12, 1.0, 1.0, 0.0, 0.0)**.

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot*(1G). See *plot*(5) for more details.

last()
in immediate mode, *last* returns the most recently computed value.

PROGRAMMING TIPS

Using *bs* as a calculator:

```
$ bs
#   Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

#   Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4  bal = bal + bal*int
bal - 1000
346.855007
...
exit
```

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while  ?(str = read)
        ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
```

```
exit
# last input line:
run
```

Input/Output examples:

```
#   Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

#   Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

SEE ALSO

ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), Section 3 of this volume for further description of the mathematical functions (pow, see exp(3), is used for exponentiation), plot(5). *Bs* uses the Standard Input/Output package.

CAL(1)

NAME

cal - print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive. Beware that "cal 78" refers to the early Christian era, not the 20th century.

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Dec. 7," "december 7," "12/7," etc., are recognized, but not "7 December' or "7/12". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in his login directory and sends him any positive results by *mail*(1). Normally this is done daily in the wee hours under control of *cron*(1M).

FILES

calendar
/usr/lib/calprog to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
/usr/lib/crontab

SEE ALSO

cron(1M), mail(1).

BUGS

Your calendar must be public information for you to get reminder service.
Calendar's extended idea of "tomorrow" does not account for holidays.

NAME

cat - concatenate and print files

SYNOPSIS

cat [-u] [-s] file ...

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output. Thus:

```
cat file
```

prints the file, and:

```
cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument - is encountered, *cat* reads from the standard input file. Output is buffered in 512-byte blocks unless the **-u** option is specified. The **-s** option makes *cat* silent about non-existent files. No input file may be the same as the output file unless it is a special file.

NOTES

Plexus provides a standalone version of *cat* in addition to the one that runs under Sys3.

SEE ALSO

cp(1), pr(1).

CB(1)

NAME

cb - C program beautifier

SYNOPSIS

cb [file]

DESCRIPTION

Cb places a copy of the C program in *file* (standard input if *file* is not given) on the standard output with spacing and indentation that displays the structure of the program.

NAME

cc, pcc - C compiler

SYNOPSIS

cc [option] ... file ...
pcc [option] ... file ...
ncc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler. *Pcc* is another name for *cc*. *Ncc* is a SYSTEM III-compatible version of the C compiler; its optimizer does a slightly better job. *Ncc* represents the latest version of *cc* and may not be available in some releases. These commands accept several types of arguments:

Arguments whose names end with **.c** are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with **.o** substituted for **.c**. The **.o** file is normally deleted, however, if a single C program is compiled and loaded all at one go.

In the same way, arguments whose names end with **.s** are taken to be assembly source programs and are assembled, producing a **.o** file.

The following options are interpreted by *cc* and *pcc*. See *ld*(1) for link editor options.

-c Suppress the link edit phase of the compilation, and force an object file to be produced even if only one program is compiled.

-p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one that automatically calls *monitor*(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1). (Z8000 only)

-O Invoke an object-code optimizer.

-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed **.s**.

-E Run only the macro preprocessor on the named C programs, and send the result to the standard output.

-P Run only the macro preprocessor on the named C programs, and leave the result on corresponding files suffixed **.i**.

-C Comments are not stripped by the macro preprocessor.

-D*name*=*def*
-D*name* Define the *name* to the preprocessor, as if by **#define**. If no definition is given, the name is defined as 1.

-U*name* Remove any initial definition of *name*.

-I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in " " will be searched for first in the directory of the *file* argument, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the *file* argument is not searched.

Other arguments are taken to be either link editor option arguments, or C-compatible object programs, typically produced by an earlier *cc* or *pcc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the

order given) to produce an executable program with the name **a.out**.

The loader (*ld*(1)) accepts 8-character symbols, and the first character of each symbol is an underbar ('_'), which *cc* prefixes at compile time. Therefore, symbol names in program modules that are to be linked must be unique within the first seven characters.

FILES

Files with '[n]' are *ncc* versions.

file.c	input file
file.o	object file
a.out	linked output
/lib/cpp	preprocessor
/usr/lib/[n]ccom	compiler, *pcc*
/lib/[n]c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/libc[n].a	standard library, see (3)
/usr/include	standard directory for **#include** files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978.
B. W. Kernighan, *Programming in C-A Tutorial.*
D. M. Ritchie, *C Reference Manual.*
adb(1), as(1), ld(1), prof(1), monitor(3C).

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor.

BUGS

If a *#define* line contains a continuation (\ newline), *cc* miscounts the number of lines in the program.

The following SYSTEM III options are not supported: -f, -g, -d, -B, and -t.

Cc does not allow more than 250 switches in a case statement. (Z8000 only)

NAME

cd - change working directory

SYNOPSIS

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 execute (search) permission in *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.

SEE ALSO

pwd(1), sh(1), chdir(2).

CDC(1)

NAME

cdc - change the delta commentary of an SCCS delta

SYNOPSIS

cdc **-r**SID [**-m**[mrlist]] [**-y**[comment]] files

DESCRIPTION

Cdc changes the *delta commentary*, for the *SID* specified by the **-r** keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (**MR**) and comment information normally specified via the *delta*(1) command (**-m** and **-y** keyletters).

If a directory is named, *cdc* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to *cdc*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

-r*SID* Used to specify the *S*CCS *ID*entification (*SID*) string of a delta for which the delta commentary is to be changed.

-m[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a list of **MR** numbers to be added and/or deleted in the delta commentary of the *SID* specified by the **-r** keyletter *may* be supplied. A null **MR** list has no effect.

MR entries are added to the list of **MR**s in the same manner as that of *delta*(1). In order to delete an **MR**, precede the **MR** number with the character **!** (see *EXAMPLES*). If the **MR** to be deleted is currently in the list of **MR**s, it is removed and changed into a "comment" line. A list of all deleted **MR**s is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If **-m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **-y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, *cdc* terminates and the delta commentary remains unchanged.

-y[*comment*] Arbitrary text used to replace the *comment*(s) already existing for the delta specified by the **-r** keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file

adds bl78-12345 and bl79-00001 to the **MR** list, removes bl77-54321 from the **MR** list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc -r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (- on the command line), then the **-m** and **-y** keyletters must also be used.

FILES

x-file (see *delta*(1))
z-file (see *delta*(1))

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

chmod - change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

The permissions of each named file are changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see *chmod*(2)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

Op can be + to add *permission* to the file's mode, - to take away *permission*, or = to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save text - sticky); **u**, **g** or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g** and **t** only works with **u**.

Only the owner of a file (or the super-user) may change its mode.

EXAMPLES

The first example denies write permission to others, the second makes a file executable:

```
chmod o-w file

chmod +x file
```

SEE ALSO

ls(1), chmod(2).

NAME

chown, chgrp - change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

Chgrp changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

FILES

/etc/passwd
/etc/group

SEE ALSO

chown(2), group(5), passwd(5).

CHROOT(1M)

NAME

chroot - change root directory for a command

SYNOPSIS

chroot newroot command

DESCRIPTION

The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

```
chroot newroot command >x
```

will create the file **x** relative to the original root, not the new one.

This command is restricted to the super-user and should begin with a slash (/).

The new root path name is always relative to the current root: even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

SEE ALSO

chdir(2).

BUGS

One should exercise extreme caution when referencing special files in the new root file system.

NAME

clear - clear terminal screen

SYNOPSIS

/usr/plx/clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type and then in */etc/termcap* to figure out how to clear the screen.

FILES

/etc/termcap terminal capability data base

NOTES

This command is based on a similar one developed at the University of California at Berkeley.

NAME

clri - clear i-node

SYNOPSIS

/etc/clri file-system i-number ...

DESCRIPTION

Clri writes zeros on the 64 bytes occupied by the i-node numbered *i-number*. *File-system* must be a special file name referring to a device containing a file system. After *clri* is executed, any blocks in the affected file will show up as "missing" in an *fsck*(1M) of the *file-system*. This command should only be used in emergencies and extreme care should be exercised.

Read and write permission is required on the specified *file-system* device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to *zap* an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M), fs(5).

BUGS

If the file is open, *clri* is likely to be ineffective.

NAME

cmp - compare two files

SYNOPSIS

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

DESCRIPTION

The two files are compared. (If *file1* is -, the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:

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

-s Print nothing for differing files; return codes only.

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

col - filter reverse line-feeds

SYNOPSIS

col [-bfpx]

DESCRIPTION

Col reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code **ESC-7**), and by forward and reverse half-line-feeds (**ESC-9** and **ESC-8**). *Col* is particularly useful for filtering multicolumn output made with the **.rt** command of *nroff*(1) and output resulting from use of the *tbl*(1) preprocessor.

If the **-b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

Although *col* accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the **-f** (fine) option; in this case, the output from *col* may contain forward half-line-feeds (**ESC-9**), but will still never contain either kind of reverse line motion.

Unless the **-x** option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters **SO** (\017) and **SI** (\016) are assumed by *col* to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output **SI** and **SO** characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, **SI**, **SO**, **VT** (\013), and **ESC** followed by **7**, **8**, or **9**. The **VT** character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, *col* will ignore any unknown to it escape sequences found in its input; the **-p** option may be used to cause *col* to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

SEE ALSO

nroff(1), tbl(1).

NOTES

The input format accepted by *col* matches the output produced by *nroff*(1) with either the **-T37** or **-Tlp** options. Use **-T37** (and the **-f** option of *col*) if the ultimate disposition of the output of *col* will be a device that can interpret half-line motions, and **-Tlp** otherwise.

BUGS

Cannot back up more than 128 lines.
Allows at most 800 characters, including backspaces, on a line.
Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comb - combine SCCS deltas

SYNOPSIS

comb [**-o**] [**-s**] [**-p**sid] [**-c**list] files

DESCRIPTION

Comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

-p*SID* The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

-c*list* A *list* (see *get*(1) for the syntax of a *list*) of deltas to be preserved. All other deltas are discarded.

-o For each **get -e** generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the **-o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

-s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original - combined) / original

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB The name of the reconstructed SCCS file.
comb????? Temporary.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

comm - select or reject lines common to two sorted files

SYNOPSIS

comm [- [**123**]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name - 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(1), diff(1), sort(1), uniq(1).

NAME

copytape - make an image copy of a tape

SYNOPSIS

/usr/plx/copytape [-rwv] [-p numfiles] [-f filenum] [-d descfile] [-i] srcfile **[-o]** dstfile

DESCRIPTION

Copytape is used for duplicating tapes. It preserves blocking and file marks. The *-r* option specifies that *srcfile* (presumably a tape) is to be read and its data placed on *dstfile*. If not otherwise specified, standard output contains the blocking and file mark information. The *-w* option (default) specifies that *srcfile* is to be read and *dstfile* (presumably a tape) is to be written according to information given as standard input.

The *-v* option (used with the *-r* option) specifies that variable size blocks may occur within a tape file.

The *-p* option must be used for the streaming tape drive, and the number of files to be read must be specified. A raw disk file system (e.g, **/dev/rdk3**), as opposed to a file, MUST be used when the streaming tape drive *-p* option is specified.

The *-f* option specifies that a single file is to be read from or written to tape. The *filenum* selects the file from the *srcfile*, starting with file number 0.

-i signals the input file, while *-o* means the output file.

EXAMPLES

The command

```
copytape -r /dev/rmt0 tapeimage > descfile
```

makes an image of the tape in drive 0 in the file *tapeimage* while creating a description file called *descfile*. By loading a new tape and issuing the command

```
copytape -w tapeimage /dev/nrmt0 < descfile
```

an exact image of the tape will be created. Notice that **/dev/nrmt0** is used instead of **/dev/rmt0**. This is required so that the tape will not rewind between files. Also notice that *tapefile* may be very large, and that there must be enough room in the file system to hold it before this will work. It is also possible to use logical disk drives (e.g., **/dev/dk5**), but this can be extremely dangerous if used incorrectly. Note that a cartridge tape will operate in streaming mode only if a raw logical disk is specified.

NOTES

This command is a Plexus feature; it is not part of standard System III.

BUGS

The *-v* option doesn't work for streaming cartridge tape drives.

NAME

cp, ln, mv - copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target
ln file1 [file2 ...] target
mv file1 [file2 ...] target

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same. If *target* is a directory, then one or more files are copied (linked, moved) to that directory.

If *mv* determines that the mode of *target* forbids writing, it will print the mode (see *chmod*(2)) and read the standard input for one line (if the standard input is a terminal); if the line begins with **y**, the move takes place; if not, *mv* exits.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent.

SEE ALSO

cpio(1), link(1M), rm(1), chmod(2).

BUGS

If *file1* and *target* 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.

Ln will not link across file systems.

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o [**acBv**]

cpio -i [**Bcdmrtuv6s**] [patterns]

cpio -p [**adlmruv**] 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.

Cpio -i (copy in) extracts from the standard input (which is assumed to be the product of a previous **cpio -o**) the names of files selected by zero or more *patterns* given in the name-generating notation of *sh*(1). In *patterns*, meta-characters **?**, *****, and **[...]** match the slash / character. The default for *patterns* is ***** (i.e., select all files).

Cpio -p (pass) copies out and in in a single operation. Destination path names are interpreted relative to the named *directory*.

When copying regular files into a directory, *cpio* normally uses the *creat*(2) system call, which preserves links to the files copied.

The meanings of the available options are:

- **a** Reset access times of input files after they have been copied.
- **B** Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from **/dev/rmt?**).
- **d** *Directories* are to be created as needed.
- **c** Write *header* information in ASCII character form for portability.
- **r** Interactively *rename* files. If the user types a null line, the file is skipped.
- **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). If the owner of the file being copied does not have write permission on the file, all links to it are removed.
- **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*(1)).
- **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.
- **6** Process an old (i.e., UNIX *Sixth* Edition format) file. Only useful with **-i** (copy in).
- **s** Swap bytes. This option cannot be used in combination with **c**.

When extracting from standard input (**-i** option) into a directory, *cpio* does not change the mode of files if they exist already in the directory. It does not change the owner or group of directories if they already exist, nor does it change a special file if it already exists.

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/mt0

cd olddir
find . -print | cpio -pdl newdir
```

The trivial case "find . -print | cpio -oB >/dev/rmt0" can be handled more efficiently by:

find . -cpio /dev/rmt0

SEE ALSO

ar(1), find(1), cpio(5).

BUGS

Path names are restricted to 128 characters.

If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost.

Only the super-user can copy special files.

When you attempt a *cpio -id* from a read-only directory, the message "Cannot chdir (no 'd' option)" is issued -- even though you have specified the 'd' option -- since you can't create directories within the original read-only directory.

NAME

crash - examine system images

SYNOPSIS

/etc/crash [system [namelist [physaddr]]]

DESCRIPTION

Crash is an interactive utility for examining an operating system core image. It has facilities for interpreting and formating the various control structures in the system and certain miscellaneous functions that are useful when perusing a dump.

The arguments to *crash* are the file name where the system image can be found, a namelist file to be used for symbol values, and the segment address of the initial process to be examined. The current process can be changed via subsequent commands. The default values are **/dev/mem**, **/sys3**, and the location of the swapper, process 0; hence, *crash* with no arguments can be used to examine an active system. If a system image file is given, it is assumed to be a system core dump and the initial process is set to be that of the process running at the time of the crash. This is determined by the value of **physaddr** stored in a fixed location by the system dump mechanism.

COMMANDS

Input to *crash* is typically of the form:

command [options] [structures to be printed].

When allowed, options will modify the format of the printout. If no specific structure elements are specified, all valid entries will be used. As an example, **proc - 12 15 3** would print process table slots 12, 15 and 3 in a long format, while **proc** would print the entire process table in the standard format. The current repertory consists of:

physaddr [physical address]
Print the location of the current process if no argument is given, or set the location to that of the supplied value.

u Print the user structure of the current process as determined by the value of *physaddr*.

trace[-r]
Generate a kernel stack trace of the current process. If the **-r** option is used, the trace begins at the saved stack frame pointer. Otherwise the trace starts at the bottom of the stack and attempts to find valid stack frames deeper in the stack.

fp [stack frame pointer]
Print the program's idea of the start of the current stack frame (set initially from a fixed location in the dump) if no argument is given, or set the frame pointer to the supplied value.

stack Format an octal dump of the kernel stack of the current process. The addresses shown are virtual system data addresses rather than true physical locations.

proc [**-[r]**] [list of process table entries]
Format the process table. The **-r** option causes only runnable processes to be printed. The **-** alone generates a longer listing.

inode [**-**] [list of inode table entries]
Format the inode table. The **-** option will also print the inode data block addresses.

file [list of file table entries]
Format the file table.

mount [list of mount table entries]

Format the mount table.

text [list of text table entries]
Format the text table.

tty [type] [-] [list of tty entries]
Print the tty structures. The *type* argument determines which structure will be used (such as **us** or **sp**; the last *type* is remembered). The - option prints the **stty** parameters for the given line.

stat Print certain statistics found in the dump. These include the panic string, time of crash, system name, and the registers saved in low memory by the dump mechanism.

var Print the tunable system parameters.

buf [list of buffer headers]
Format the system buffer headers.

buffer [format] [list of buffers]
Print the data in a system buffer according to *format*. Valid formats include **hex**, **decimal**, **octal**, **character**, **byte**, **directory**, **inode**, and **write**. The last creates a file containing the buffer data.

callout Print all entries in the callout table.

map list of map names
Format the named system map structures.

nm list of symbols
Print symbol value and type as found in the namelist file.

ts list of text addresses
Find the closest text symbols to the given addresses.

ds list of data addresses
Find the closest data symbols to the given addresses.

od symbol or data address [count [format]]
Dump *count* data values starting at the symbol value or address given according to *format*. Allowable formats are **hex**, **octal**, **decimal**, **character**, or **byte**.

! Escape to shell.

q Exit from *crash*.

? Print synopsis of commands.

ALIASES

There are built in aliases for many of the commands and formats. In general, the first letter of a name is satisfactory, thus, **k** is a shorthand notation for **kernel**. Exceptions are **x** for **text** and **e** for **decimal**.

FILES

/dev/mem	default system image file
/sys3	default namelist file
buf.#	files created containing buffer data

NOTES

This program has been changed to reflect Plexus hardware.

SEE ALSO

crash(8).

BUGS

This program is not completely implemented on Plexus hardware. Specifically, the *physaddr*, *u*, *trace*, *fp*, *stack*, and *tty* commands are either missing or incomplete. Not all commands print in hexadecimal.

NAME

cref - make cross-reference listing

SYNOPSIS

cref [**-acilnostux123**] files

DESCRIPTION

Cref makes a cross-reference listing of assembler or C programs; *files* are searched for symbols in the appropriate syntax.

The output report is in four columns:

1. symbol;
2. file name;
3. see below;
4. text as it appears in the file.

Cref uses either an *ignore* file or an *only* file. If the **-i** option is given, the next argument is taken to be an *ignore* file; if the **-o** option is given, the next argument is taken to be an *only* file. *Ignore* and *only* files are lists of symbols separated by new-lines. All symbols in an *ignore* file are ignored in columns 1 and 3 of the output. If an *only* file is given, only symbols in that file will appear in column 1. Only one of these options may be given; the default setting is **-i** using the default ignore file (see *FILES* below). Assembler pre-defined symbols or C keywords are ignored.

The **-s** option causes current symbols to be put in column 3. In the assembler, the current symbol is the most recent name symbol; in C, the current function name. The **-l** option causes the line number within the file to be put in column 3.

The **-t** option causes the next available argument to be used as the name of the intermediate file (instead of the temporary file **/tmp/crt??**). This file is created and is *not* removed at the end of the process.

The *cref* options are:

a	assembler format (default)
c	C format input
i	use an *ignore* file (see above)
l	put line number in column 3 (instead of current symbol)
n	omit column 4 (no context)
o	use an *only* file (see above)
s	current symbol in column 3 (default)
t	user-supplied temporary file
u	print only symbols that occur exactly once
x	print only C external symbols
1	sort output on column 1 (default)
2	sort output on column 2
3	sort output on column 3.

FILES

/tmp/crt??	temporaries
/usr/lib/cref/aign	default assembler *ignore* file
/usr/lib/cref/atab	grammar table for assembler files
/usr/lib/cref/cign	default C *ignore* file
/usr/lib/cref/ctab	grammar table for C files
/usr/lib/cref/crpost	post-processor
/usr/lib/cref/upost	post-processor for **-u** option

SEE ALSO

as(1), cc(1), sort(1), xref(1).

BUGS

Cref inserts an ASCII DEL character into the intermediate file after the eighth character of each name that is eight or more characters long in the source file.

NAME

cron - clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file **/usr/lib/crontab**. Because *cron* never exits, it should be executed only once. This is best done by running *cron* from the initialization process through the file **/etc/rc** (see *init*(8)).

The file **crontab** consists 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); or
an asterisk (meaning all legal values).

"0" is a valid entry for the day-of-month and month-of-year fields, even though it is out of range for these fields. It means "never" and can be used to turn off processes.

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 new-line character. Only the first line (up to a **%** or the end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

Cron examines **crontab** once a minute to see if it has changed; if it has, *cron* reads it. Thus it takes only a minute for entries to become effective.

Cron ignores a line in **crontab** if one of the fields has an invalid value.

FILES

/usr/lib/crontab
/usr/lib/cronlog

SEE ALSO

sh(1), init(8).

DIAGNOSTICS

A history of all actions by *cron* are recorded in **/usr/lib/cronlog**.

BUGS

Cron reads **crontab** only when it has changed, but it reads the in-core version of that table once a minute. A more efficient algorithm could be used. The overhead in running *cron* is about one percent of the CPU, exclusive of any commands executed by *cron*.
Changing the date by a few hours or more can cause *cron* to execute commands like crazy trying to catch up. This can load down and even harm your system if *cron* tries to run incompatible processes at once.

NAME

crypt - encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

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

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* 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.

Crypt 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.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) 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(1), makekey(8).

BUGS

If output is piped to *nroff*(1) and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)).

NAME

csh - a shell (command interpreter) with C-like syntax

SYNOPSIS

/usr/plx/csh [**-cefinstvVxX**] [arg ...]

DESCRIPTION

Csh is a command language interpreter. When you invoke it, it first executes commands from the file **.cshrc** in your *home* directory. If you are logging in, it also executes commands from the file **.login** there. Normally the shell then begins reading commands from the terminal, prompting with **%** (a per cent sign followed by a blank). Later in this manual entry, we will describe how the shell processes arguments and command scripts.

The shell repeatedly reads a line of command input and breaks the line into *words;* places the sequence of words on the command history list and parses it; and finally executes each command in the current line.

When a login C-shell terminates, it executes commands from the file **.logout** in your *home* directory.

Lexical Structure

The shell usually splits input lines into words at blanks and tabs. The characters &, |, ;, <, >, (, and) are exceptions, however; they all form separate words. If doubled, as in &&, ||, <<, or >>, these pairs form single words. These parser metacharacters may be made part of other words; their special meaning may be turned off 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. The semantics of these quotations are described below. 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 is prevented this special meaning when preceded by \ and in quotations using `, ', and ".

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. In a pipeline, the output of each command becomes the input of the next. A command line may contain sequences of pipelines; separate the pipelines by ;, and they are then executed sequentially. You do not necessarily have to wait for a sequence of pipelines to finish executing before you issue another command: by following the command with an ampersand (&), the sequence of pipelines (0 or more) is executed in background mode, and you receive another shell prompt immediately. A command sequence followed by an ampersand is not terminated by a hangup signal; the *nohup* command need not be used.

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

Substitutions

The shell performs various transformations on its input.

History Substitutions

History substitutions reintroduce sequences of words from previous commands. They may also perform 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. This *!* may be preceded by an **** to prevent its special meaning; a *!* is passed unchanged when it is followed by a blank, tab, newline, = or *(*. History substitutions also occur when an input line begins with ↑. This special abbreviation is described later.

Input lines containing history substitution metacharacters are echoed on the terminal before being executed. The echoed version shows the command line as it could have been typed without history substitution.

The history mechanism saves some number of commands input from the terminal. The size of the *history list* thus created is controlled by the *history* variable. The immediately previous command is always retained. Commands are numbered sequentially from 1.

For example,, 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. You don't always need to use event numbers when you use the history mechanism; if you want to see them for each command, the current event number can be made part of the *prompt* by placing an **!** in the prompt string.

If the current event is number 13, you can refer to event 11 by its event number as in **!11**; by its event number relative to the current event number as in **!-2**; by a prefix of a command word as in **!c**; or by a string contained in a word in the command as in **!?old?**. These forms, without further modification, simply repeat the command line of event 11. As a special case **!!** refers to the previous command; thus **!!** alone is essentially a *redo.* The form **!#** refers to the current command (the one being typed in). See below for an example of this in use.

To select words from a previous command line, use a colon (:) and a designator for the desired words. The words of a input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. 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 "$"

Thus the command

diff /usr/man/docs/vpm1.0 /usr/man/docs/vpm2.0 ; vi !#:1

uses both the "!#" convention for the current command line, and the "*:n*" convention for argument number. The effect of this command is to display the differences between the two files on the standard output, and then summon the editor **vi** for use on the first file (argument number 1).

The **:** separating the event specification from the word designator can be omitted if the argument selector begins with a ↑, **$**, *****, **-**, or **%**. So, in the example above, the **vi** portion could have been equivalently typed

vi !#^

A sequence of modifiers can be placed after the optional word designator. Each modifier is preceded by a :. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing *.xxx* component, leaving the root name.
s/*l*/*r*/	Substitute *l* for *r*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. *g&*.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a *g* the modification is applied only to the first modifiable word. It is always an error for no word to be applicable.

Expressions on the left hand side of substitutions are not regular expressions in the sense of the editors; rather, they are strings. Any character may be used as the delimiter in place of /; a **** quotes the delimiter. The character **&** in the right hand side is replaced by the text from the left. A **** quotes **&** also. A null *l* (left hand side expression) 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; the same goes for the trailing **?** in a contextual scan.

You can use a history reference 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 non-blank 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 **}** to insulate it from the characters that follow. Thus, after **ls ~sandy** we might do **!{l}1** to get **ls ~sandy1**, while **!l1** would look for a command starting **l1**.

Quotations with ' and "

The quotation of strings by ' and " can prevent all or some substitutions. Strings enclosed in ' are prevented any further interpretation. Strings enclosed in " are variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitition* 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 that can be established, displayed and modified by the *alias* and *unalias* commands. After scanning a command line, the C shell parses it into distinct commands and checks the first word of each command, left-to-right, to see if it has an alias. If it does, then the text that is the alias for that command is reread as though that command were the previous input line. The history mechanism remains fully operational within aliasing. The resulting words replace the command and argument list.

Thus if the alias for **ls** is **ls -l**, the command **ls /usr** becomes **ls -l /usr**. The argument list here is undisturbed. Similarly if the alias for **lookup** is **grep !↑ /etc/passwd**, then **lookup bill** becomes **grep bill /etc/passwd**.

If the C shell finds an alias, it transforms the words of the input text and begins the aliasing process again on the reformed input line. If the first word of the new text is the same as the old, the shell flags 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 \f1pr \!* | lpr\f1** to make a command that *pr*s its arguments to the line printer.

Variable Substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. 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. For a complete list of the shell's pre-defined variables, see the section *Pre-defined Variables* towards the end of this manual entry.

The values of variables may be displayed and changed by using the *set* and *unset* commands. A number of the variables referred to by the shell 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 that causes command input to be echoed. The setting of this variable results from the *-v* command line option.

Other operations treat variables numerically. The @ command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, 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 alaising and parsing the input line, and before executing each command, the shell performs variable substitution, keyed by **$** characters. This expansion can be prevented by preceding the **$** with a ****, except within **always** occurs, and within **'fRs where it never** occurs. Strings quoted by ` are interpreted later (see *Command Substitution* below) so **$** substitution does not occur there until later, if at all. A **$** is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable-expanded separately. With no I/O redirection, the command name and entire argument list are expanded together. Thus the first (command) word may generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in " or given the **:q** modifier, the results of variable substitution may eventually be command and filename substituted. Within ", 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 metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to refer to a variable that 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 that 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 variable's 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, the modifiers must appear within the braces. The current implementation allows only one : modifier 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.

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

Command and Filename Substitution

Command and filename substitutions are applied selectively to the arguments of built-in commands. This means that portions of expressions that are not evaluated are not subjected to these expansions. For commands 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

Enclosing a command in ` indicates command substitution. The shell breaks the output from such a command into separate words at blanks, tabs and newlines; it discards null words. It then replaces the original string with this text. Within "s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Thus a command substitution may 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 replaced with an alphabetically sorted list of file names that match the pattern. If several words on the same command line specify filename substitution, the C shell returns an error only if no pattern matches an existing file name. It does not return an error if some matches are successful and others are not. Only the metacharacters *, **?** and **[** imply pattern matching; the characters **~** and **{** being 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 **[...]** matches any one of the characters enclosed. Within **[...]**, a pair of characters separated by - matches any character lexically between the two.

The character ~ at the beginning of a filename refers to home directories. Standing alone, i.e. ~, it expands to the invoker's home directory as reflected in the value of 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 his 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 appears someplace other than at the beginning of a word, it is left undisturbed.

The metanotation **a{b,c,d}e** is a shorthand for **abe ace ade**. The shell preserves left to right order, and sorts the results of matches separately at a low level to preserve this order. This construct may be nested. Thus, **~source/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/ls.c**. This works whether or not these files exist. There is no chance of error if the home directory for **source** is **/usr/source**. Similarly **../{memo,*box}** might expand to **../memo ../box ../mbox**. (Note that **memo** is not sorted with the results of matching ***box**.) As a special case **{**, **}** and **{}** are passed undisturbed.

Input/Output

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

< name
: Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word
: Read the shell input up to a line identical to *word.* The shell does not perform variable, filename or command substitution on *word.* It compares each input line to *word* before doing any substitutions on this input line. Unless a quoting \, ", ' or ` appears in *word,* the shell performs variable and command substitution on the intervening lines, allowing \ to quote $, \ and `. Commands that are substituted have all blanks, tabs, and newlines preserved, except for the final newline, which is dropped. The shell places the resultant text in an anonymous temporary file, which it then gives to the command as standard input.

> name
>! name
>& name
>&! name
: The file *name* is used as standard output. If the file does not exist, it is created; if the file exists, its previous contents are lost.

  The variable *noclobber* is designed to prevent accidental overwriting of files by ">". If the variable *noclobber* is set, then the file named by *name* must either not exist or be a character special file (e.g. a terminal or */dev/null*); otherwise, *noclobber* prevents the redirection and issues an error message. The **!** forms 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 simple command is run in foreground mode, without being followed by &, its environment is that of the shell that runs it. If a shell procedure is run in foreground mode, without being followed by &, each command within the procedure receives the environment in which the procedure was invoked. In other words, the shell spawned by the shell procedure inherits the environment of the shell that spawned it. This environment may be modified by command-line factors such as input-output parameters or the presence of the command in a pipeline. Thus commands run from within a shell procedure receive the standard input of the shell that is running the script; commands within a shell script know nothing about each other. Since we often want a command within a shell script to receive standard input *not* from the shell that runs the script but from within the script itself, we need a way to present such inline data. The << mechanism serves this function. It permits shell command scripts to function as components of pipelines and allows the shell to block read its input. See *An Introduction to the C Shell* for examples of the use of the << mechanism.

If a command or shell procedure is run detached (followed by &), its default standard input is the empty file */dev/null*.

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

Expressions

A number of the shell's built-in commands (described in the section "Built-in Commands" below) take *expressions*, in which the operators are similar to those of C, with the same precedence. Built-in commands that take expressions include @, *exit, if,* and *while.* The following operators are available for use within expressions:

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

In this list the precedence increases to the right, and down, == 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 that begin with 0 are considered octal numbers. The shell evaluates null or missing arguments as 0. The results of all expressions are strings, which represent decimal numbers. Components of expressions should be surrounded by spaces; this always matters, except when components are adjacent to &, |, <, >, (, or), which are syntactically significant to the parser.

Command executions enclosed in { and } and file enquiries are also available in expressions as primitive operands. File enquiries take 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

"Name" is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, 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 you want more detailed status information about a command, execute the command outside of an expression and examine the variable *status.*

Control Flow

The shell contains commands that can regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input. Due to the implementation, the shell restricts the placement of some of the commands.

The *foreach, switch,* and *while* statements, as well as the *if-then-else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

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*'s succeed on non-seekable 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, 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 substituted. *Name* may not be *alias* or *unalias*

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

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

case label:
A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name
Change the shell's working directory to directory *name.* If no argument is given then change 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 ../), 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.

continue
Continue 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, akin to the **\c** in *nroff*(1). A **\n** in wordlist causes a newline to be printed.

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* and *end* must appear alone on separate lines.)

The built-in command *continue* continues the loop prematurely and the built-in command *break* terminates it prematurely. When the C shell reads a *foreach* command from the terminal, it reads the loop once and prompts with *?* before executing any statements in the loop. If you make a mistake typing in a loop at the terminal, you can rub it out.

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

goto word
The specified *word* is filename and command expanded 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 *label* 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 as 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, when command is *not* executed (this is a bug).

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif
If the specified *expr* is true then all the commands up to the first *else* are executed; if *expr2* is true then the commands to 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* must appear alone on its input line or after an *else.)*

login

Terminate a login shell, replacing it with an instance of **/bin/login.** This is one way to log off, included for compatibility with **/bin/sh.**

logout

Terminate a login shell. Especially useful if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command

The first form sets the *nice* for this shell to 4. 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 ....** Command is always executed in a sub-shell, and the restrictions on commands in simple *if* statements apply.

nohup
nohup command

Shell scripts use the first form 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 detached, *nohup* has no effect. All processes detached with & are automatically *nohup*ed. (Thus, *nohup* is not really needed.)

The SYSTEM III utility **/bin/nohup** is incompatible with the *csh nohup*. Therefore, to use the SYSTEM III command, you must invoke it with its full pathname.

onintr
onintr -
onintr label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts (i.e., to terminate shell scripts or 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 detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

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. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* (which is subject to the same restrictions as the *command* in the one-line *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 that 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 *index*th component of *name* to *word;* this component must already exist. The final form sets *name* to the list of words in *wordlist.* In all cases the value is command and filename expanded.

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

(Version 7 systems only.) Sets the value of environment variable *name* to be *value,* a single string. Useful environment variables are *TERM*, the type of your terminal, and *SHELL*, the shell you are using.

shift
shift variable

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 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; 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. Input during *source* commands is *never* placed on the history list.

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

Each *case* label is successively matched against the specified *string,* which is first command and filename expanded. The file metacharacters *, **?** and **[...]** may be used in the *case* labels, which are variable expanded. 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, the shell prints a summary of time used by this shell and its children. With arguments, the shell times the specified simple command and prints a time summary as described under the *time* variable. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask value

The file creation mask is displayed (first form) or set to the specified value (second form). 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 all access except no write access for users in the group or others.

unalias pattern
The shell discards all aliases whose names match the specified pattern. 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
The shell removes all variables whose names match the specified pattern. Thus all variables are removed by *unset* *; this can have distasteful side-effects. It is not an error for nothing to be *unset.*

wait
The shell waits for all child processes. 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 non-zero, the commands between the *while* and the matching *end* are evaluated. *Break* and *continue* can terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) If the input is from a terminal, prompting occurs here the first time through the loop as for the *foreach* statement.

@
@ 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 parentheses ( ). 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, however, mandatory in separating components of *expr* that would otherwise be single words.

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

Pre-defined Variables

The following variables have special meaning to the shell. Of these, the shell always sets *argv, child, home, path, prompt, shell* and *status.* This setting occurs only at initialization, except for *child* and *status.* Variables set by the shell are not subsequently modified by the shell, though the user may explicitly modify them.

The shell copies the environment variable PATH into the variable *path*, and copies the value back into the environment whenever *path* is set. Thus you need not worry about its setting other than in the file *.cshrc,* because inferior *csh* processes 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.

cdpath Specifies a list of alternate directories to be searched by *chdir* commands.

child The process number printed when the last command was 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 non-built-in commands, all expansions occur before echoing. Builtin 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=",;"* makes the history characters a comma and semicolon.

history Takes a numeric value that controls the size of the history list. The shell does not discard any command referenced in this many events. Too large values of *history* 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 shells from accidentally being killed by control-Ds.

mail The files where the shell checks for mail. The shell checks for mail after each command completion that results in a prompt, if a specified interval has elapsed. If any of these files exists with an access time not greater than its modify time, the shell sends the message "You have new mail."

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

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

noclobber As described in the section on *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 that do not deal 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 no *path* variable is specified, only full path names work. The usual search path is *.*, */bin* and */usr/bin*, but this may vary from system to system. For the super-user the default search path is */etc*, */bin* and */usr/bin*. A shell that 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* or the commands may not be found.

prompt The string printed before each command is read from an interactive terminal input. If a *!* appears in the string it is 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 *Non-built-in Command Execution* below.) Initialized to the (system-dependent) 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 that fail return exit status 1, all other built-in commands set status 0.

time Controls automatic timing of commands. *Time* takes a numeric argument, which stands for some number of CPU seconds. If *time* is set, the shell prints a line of information when any command taking more than this many CPU seconds terminates. The line gives user, system, and real times, 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.

Non-built-in Command Execution

When the shell finds that a command to be executed is not a built-in command, it tries to execute the command via *exec*(2). Each word in the variable *path* names a directory from which the shell attempts to execute the command. If it is given neither a **-c** nor a **-t** option, the shell hashes the names in these directories into an internal table so that it only tries an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when the search path contains a large number of directories. If this mechanism has been turned off (via *unhash)*, or if the shell is given a **-c** or **-t** argument, and in any case for each directory component of *path* that does not begin with a /, the shell tries to concatenates all the *path* entries with the given command name to form a path name of a file, which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus *(cd ; pwd) ; pwd* does not leave you in the *home* directory; it leaves you where you are, and prints the *home* directory name followed by the name of the directory you are in. *cd ; pwd*, on the other hand, leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

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 path name 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 shell 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 non-zero exit status.

-f The shell starts faster, because it neither searches for nor executes 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 not to be a terminal. Shells are interactive without this option if their inputs and outputs 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 \ escapes the newline at the end of this line to continue onto another line.

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

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

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

-X Is to **-x** as **-V** is to **-v.**

After processing of flag arguments, if arguments remain but none of the **-c, -i, -s,** or **-t** options is 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**. Many system shell procedures are written for use with either the standard Sys3 or Version 7 shells, whose shell scripts are not compatible with this shell. Therefore, the C shell executes such a *standard* shell if the first character of a 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. It ignores *interrupt* signals as well if the command is followed by &; otherwise the signals have the values the shell inherited from its parent. The shell's handling of interrupts can be controlled by *onintr.* Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. No interrupts are allowed when a login shell is reading the file *.logout.*

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after *.cshrc* at login.
~/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell 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 characters in an argument varies from system to system. The number of arguments to a command involving filename expansion is limited to 1/6th the number of characters allowed in an argument list. 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* substititutions on a single line to 20.

NOTES

The Plexus version of the C Shell is based on the one from the University of California at Berkeley.

SEE ALSO

access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(5), environ(5), *An Introduction to the C Shell.*

BUGS

Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

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

It should be possible to use the : modifiers on the output of command substitutions. More than one : modifier should be allowed on $ substitutions.

Some commands should not touch *status* or it may be so transient as to be almost useless. Or-ing in 0200 to *status* on abnormal termination is not elegant.

The new *exec* command inherits several open files other than the normal standard input and output and diagnostic output. If the input and output are redirected and the new command does not close these files, some files may be held open unnecessarily.

A number of bugs are associated with the importing/exporting of the PATH. For example, directories in the path using the ~ syntax are not expanded in the PATH. Unusual paths, such as (), can cause *csh* to dump core.

This version of *csh* does not support or use the process control features of the 4th Berkeley Distribution. It contains a number of known bugs that have been fixed in the process control version.

NAME

csplit - context split

SYNOPSIS

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 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 up 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 up 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 quotes. Regular expressions may not contain embedded new-lines. *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}
```

CSPLIT(1)

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(1), sh(1), regexp(7).

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.

NAME

ct - call terminal

SYNOPSIS

ct [**-h**] [**-v**] [**-wn**] [**-s**speed] telno

DESCRIPTION

Ct dials the phone number of a modem that is attached to a terminal, and spawns a *login* process to that terminal. *Telno* is the telephone number, with minus signs at appropriate places for delays.

Ct determines which dialers are associated with lines that are set to the appropriate speed by examining the file **/usr/lib/uucp/L-devices**. If all such available dialers are busy, *ct* will ask if it should wait for a line, and if so, for how many minutes it should wait before it gives up. *Ct* will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the **-w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

Normally, *ct* will hang up the current line, so that that line can answer the incoming call. The **-h** option will prevent this action. If the **-v** option is used, *ct* will send a running narrative to standard error.

The data rate may be set with the **-s** option, where *speed* is expressed in baud. The default rate is 300.

The destination terminal must be attached to a modem that can answer the telephone.

FILES

/usr/lib/uucp/L-devices

SEE ALSO

cu(1C), login(1), uucp(1C), dn(4), getty(8).

NAME

ctags - create a tags file

SYNOPSIS

/usr/plx/ctags [**-u**] [**-w**] [**-x**] name ...

DESCRIPTION

Ctags makes a tags file for *ex*(1) from the specified C, Pascal and Fortran sources. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition. These are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these function definitions.

If the **-x** flag is given, *ctags* produces a list of function names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

Files whose name ends in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:

-w suppressing warning diagnostics.

-u causing the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way that is rather slow; it is usually faster to simply rebuild the *tags* file.)

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

FILES

tags	output tags file

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

ex(1), vi(1).

BUGS

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name, it won't work.

The method of deciding whether to look for C or Pascal and FORTRAN functions is not very sophisticated.

NAME

cu - call another UNIX system

SYNOPSIS

cu [**-s**speed] [**-a**acu] [**-l**line] [**-h**] [**-o**|**-e**] telno | **dir**

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. *Speed* gives the transmission speed (110, 150, 300, 1200, 4800, 9600); 300 is the default value. Most of our modems restrict us to choose between 300 and 1200. Directly connected lines may be set to other speeds.

The **-a** and **-l** values may be used to specify device names for the Automatic Call Unit (ACU) and communications line devices. They can be used to override searching for the first available ACU with the right speed. The **-h** option emulates local echo, supporting calls to other computer systems which expect terminals to be in half-duplex mode. The **-e** (**-o**) option designates that even (odd) parity is to be generated for data sent to the remote. *Telno* is the telephone number, with equal signs for secondary dial tone or minus signs for delays, at appropriate places. The string **dir** for *telno* must be used for directly connected lines, and implies a null ACU.

Cu will try each line listed in the file **/usr/lib/uucp/L-devices** until it finds an available line with appropriate attributes or runs out of entries. 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.

If the ACU specified in **/usr/lib/uucp/L-devices** is not a special file, it is executed by *cu* with the *line*, *speed*, and *telno* as arguments. This feature can be used to initiate a calling sequence on most auto-dial modems. A return value of 0 means the call was initiated successfully.

The *transmit* process interprets the following:

~.	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.
~%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.
~~...	send the line ~... to the remote system.
~%nostop	turn off the DC3/DC1 input control protocol for the remainder of the session. 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. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

~>[>]: *file*
zero or more lines to be written to *file*
~>

Data from the remote is diverted (or appended, if >> is used) to file. The trailing ~> terminates the diversion.

CU(1C)

The use of **~%put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of *echo*(1) and *cat*(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

FILES

/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null

NOTES

Executing the ACU file specified in **/usr/lib/uucp/L-devices** is not a standard feature and may not be available on other UNIX systems.

Users who write their own dialer program must link (**ld**(1)) it with the "-N" option (no shared text).

SEE ALSO

cat(1), echo(1), stty(1), uucp(1C), tty(4).

D. A. Nowitz, *UUCP Implementation Description*.
This document, in Volume 2B of the *Plexus Sys3 UNIX Programmer's Manual*, describes how to set up the file **/usr/lib/uucp/L-devices**.

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

BUGS

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

If the transported file does not end in a newline character, *cu* appears to hang; typing a control-D completes the file transporting and restores the user's terminal to its normal state. Typing a delete character instead of control-D leaves the user's terminal in a weird mode.

NAME

cut - cut out selected fields of each line of a file

SYNOPSIS

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; in data base parlance, it implements the projection of a relation. 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 integer field numbers (in increasing order), with optional **-** to indicate 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 pass 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 Suppresses lines with no delimiter characters in case of **-f** option. Unless specified, lines with no delimiters will be passed through untouched.

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

HINTS

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLES

cut -d: -f1,5 /etc/passwd mapping of user IDs to names

name=`who am i | cut -f1 -d" "` to set **name** to current login name.

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.

SEE ALSO

grep(1), paste(1).

NAME

cw, checkcw - prepare constant-width text for troff

SYNOPSIS

cw [-lxx] [-rxx] [-fn] [-t] [+t] [-d] [files]

checkcw [-lxx] [-rxx] files

DESCRIPTION

Cw is a preprocessor for *troff*(1) input files that contain text to be typeset in the constant-width (CW) font.

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. (An earlier version of this font was used in typesetting *The C Programming Language* by B. W. Kernighan and D. M. Ritchie). It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

Because the CW font contains a "non-standard" set of characters and because text typeset with it requires different character and inter-word spacing than is used for "standard" fonts, documents that use the CW font must be preprocessed by *cw*.

The CW font contains the 94 printing ASCII characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
! $ & ( ) ' ' * + @ . , / : ; = ? [ ] | - _ ^ ~ " <> { } # \
```

plus eight non-ASCII characters represented by four-character *troff*(1) names (in some cases attaching these names to "non-standard" graphics), as follows:

Character	*Symbol*	*Troff Name*
"Cents" sign	¢	(ct
EBCDIC "not" sign	¬	(no
Left arrow	←	(<-
Right arrow	→	(->
Down arrow	↓	(da
Vertical single quote	'	(fm
Control-shift indicator	†	(dg
Visible space indicator	□	(sq
Hyphen	-	(hy

The hyphen is a synonym for the unadorned minus sign (-). Certain versions of *cw* recognize two additional names: \(ua for an up arrow and \(lh for a diagonal left-up (home) arrow.

Cw recognizes five request lines, as well as user-defined delimiters. The request lines look like *troff*(1) macro requests, and are copied in their entirety by *cw* onto its output; thus, they can be defined *by the user* as *troff*(1) macros; in fact, the .CW and .CN macros *should* be so defined (see *HINTS* below).

The five requests are:

.CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the *cw* command line.

.CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the *cw* command line.

.CD Change delimiters and/or settings of other options; takes the same options as are available on the *cw* command line.

.CP *arg1 arg2 arg3 . . . argn*
All the arguments (which are delimited like *troff*(1) macro arguments) are concatenated, with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

.PC *arg1 arg2 arg3 . . . argn*
Same as .CP, except that the even-numbered (rather than odd-numbered) arguments are set in the CW font.

The .CW and .CN requests are meant to bracket text (e.g., a program fragment) that is to be typeset in the CW font "as is." Normally, *cw* operates in the *transparent* mode. In that mode, except for the .CD request and the nine special four-character names listed in the table above, every character between .CW and .CN request lines stands for itself. In particular, *cw* arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes () and ligatures (fi, ff, etc.) everywhere to be "hidden" from *troff*(1). The transparent mode can be turned off (see below), in which case normal *troff*(1) rules apply. In any case, *cw* hides from the *user* the effect of the font changes generated by the .CW and .CN requests.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

The user can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text (see the example under *BUGS* below). *Cw* treats text enclosed by delimiters in precisely the same manner as text bracketed by .CW/.CN pairs, except that, for aesthetic reasons, spaces in text bracketed by .CW/.CN pairs have the same width as any other CW character, while spaces between delimiters are half as wide, so that they have the same width as spaces in the prevailing text (but are *not* adjustable).

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

-l*xx* The one- or two-character string *xx* becomes the left delimiter; if *xx* is omitted, the left delimiter becomes undefined, which it is initially.

-r*xx* Same for the right delimiter. The left and right delimiters may (but need not) be different.

-f*n* The CW font is mounted in font position *n*; acceptable values for *n* are 1, 2, and 3 (default is 3, replacing the bold font). This option is only useful at the beginning of a document.

-t Turn transparent mode *off*.

+t Turn transparent mode *on* (this is the initial default).

-d Print current option settings on file descriptor 2 in the form of *troff*(1) comment lines. This option is meant for debugging.

Cw reads the standard input when no *files* are specified, so it can be used as a filter. Typical usage is:

cw *files* | troff ...

Checkcw checks that left and right delimiters, as well as the .CW/.CN pairs, are properly balanced. It prints out all offending lines.

HINTS

Typical definitions of the .CW and .CN macros meant to be used with the *mm*(7) macro package:

```
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ...
..
.de CN
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *troff*(1) no-fill (.nf) mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point *smaller* than the prevailing point size (the displayed definitions of .CW and .CN above are one point smaller than the running text on this page). The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: *cw*, *tbl*, and *eqn*. Usually, the tables contained in such documents will not contain any CW text, although it is entirely possible to have *elements* of the table set in the CW font; of course, care must be taken that *tbl*(1) format information not be modified by *cw*. Attempts to set equations in the CW font are not likely to be either pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting ← represent a backspace, d←←dg yields đ. Because spaces (and, therefore backspaces) are half as wide between delimiters as inside .CW/.CN pairs (see above), two backspaces are required for each overstrike between delimiters.

FILES

/usr/lib/font/ftCW CW font-width table

SEE ALSO

eqn(1), mmt(1), tbl(1), troff(1), mm(7), mv(7).

WARNINGS

If text preprocessed by *cw* is to make any sense, it must be set on a typesetter equipped with the CW font or on the MHCC STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

BUGS

Only a masochist would use periods (.) or backslashes () as delimiters.
Certain CW characters don't concatenate gracefully with certain Times Roman characters, e.g., a CW ampersand (&) followed by a Times Roman comma(,); in such cases, judicious use of *troff*(1) half- and quarter-spaces (and) is most salutary, e.g., one should use _&_, (rather than just plain _&_,) to obtain &, (assuming that _ is used for both delimiters).
Using *cw* with *nroff* is silly.
The output of *cw* is hard to read.
See also *BUGS* under *troff*(1).

DATE(1)

NAME

date - print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. 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. 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*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert 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 week - Sunday = 0
a	abbreviated weekday - Sun to Sat
h	abbreviated month - Jan to Dec
r	time in AM/PM notation

EXAMPLE

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

would generate as output:

DATE: 08/01/76
TIME: 14:45:05

DIAGNOSTICS

No permission	if you aren't the super-user and you try to change the date;
bad conversion	if the date set is syntactically incorrect;
bad format character	if the field descriptor is not recognizable.

FILES

/dev/kmem

NAME

dc - desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one 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; the result is 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 a UNIX 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.

; : are 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(1), which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

x is unimplemented
where *x* is an octal number.

stack empty
for not enough elements on the stack to do what was asked.

Out of space
when the free list is exhausted (too many digits).

Out of headers
for too many numbers being kept around.

Out of pushdown
for too many items on the stack.

Nesting Depth
for too many levels of nested execution.

NAME

dd - convert and copy a file

SYNOPSIS

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	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=**ascii**	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
..., ...	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.

EXAMPLE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file **x**:

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

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

The command

```
dd if=/dev/rmt0 conv=swab bs=20b | tar xvf ...
```

is used to read *tar* format files from machines (e.g., DEC machines) that are byte-swapped with respect to the Z8000.

NOTES

Plexus provides a standalone version of *dd* in addition to the one that runs under Sys3.

SEE ALSO

cp(1).

DIAGNOSTICS

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

BUGS

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

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

The *swab* option cannot be used in the same *dd* command with either *ibm* or *ebcdic*.

NAME

delta – make a delta (change) to an SCCS file

SYNOPSIS

delta [**–r**SID] [**–s**] [**–n**] [**–g**list] [**–m**[mrlist]] [**–y**[comment]] [**–p**] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see **–m** and **–y** keyletters below).

Keyletter arguments apply independently to each named file.

–r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *get*s for editing (**get –e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **–r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

–s Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

–n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

–g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

–m[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (**MR**) number *must* be supplied as the reason for creating the new delta.

If **–m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **–y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from **MR** number validation program, *delta* terminates (it is assumed that the **MR** numbers were not all valid).

–y[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*-file are explained in the *Source Code Control System User's Guide*. The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

WARNINGS

Lines beginning with an **SOH** ASCII character (binary 001) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get*/*delta* sequences should be used.

If the standard input (**-**) is specified on the *delta* command line, the **-m** (if necessary) and **-y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

SEE ALSO

admin(1), bdiff(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

DEROFF(1)

NAME

deroff - remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [**-w**] [**-m**x] [files]

DESCRIPTION

Deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl*(1) descriptions, and writes the remainder of the file on the standard output. *Deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **-m** option may be followed by an **m**, **s**, or **l**. The resulting **-mm** or **-ms** option causes the **mm** or **ms** macros to be interpreted so that only running text is output (i.e., no text from macro lines.) The **-ml** option forces the **-mm** option and also causes deletion of lists associated with the **mm** macros.

If the **-w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (**&**), and apostrophes ('); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

SEE ALSO

eqn(1), tbl(1), troff(1).

BUGS

Deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. In particular, the output of **.tl** requests is a bit bizarre. Most such errors result in too much rather than too little output.

The **-ml** option does not handle nested lists correctly.

NAME

devnm - device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

Devnm identifies the special file associated with the mounted file system where the argument *name* resides. *name* must be a full path name.

This command is most commonly used by **/etc/rc** (see *rc*(8)) to construct a mount table entry for the **root** device.

EXAMPLE

The command:

```
/etc/devnm /usr
```

produces

```
rp1 /usr
```

if **/usr** is mounted on **/dev/rp1**.

FILES

/dev/rp*
/etc/mnttab

SEE ALSO

setmnt(1M).

NAME

df - report number of free disk blocks

SYNOPSIS

df [**-t**] [**-f**] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/rp1**) or by mounted directory name (e.g., **/usr**). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed. Blocks are 1024 bytes long.

The **-t** flag causes the total allocated block figures to be reported as well.

If the **-f** flag is given, only an actual count of the blocks in the free list is made (free i-nodes are not reported). With this option, *df* will report on raw devices.

FILES

/dev/rf*
/dev/rk*
/dev/rp*
/etc/mnttab

NOTES

Blocks are 1024 bytes long.

SEE ALSO

fsck(1M), fs(5), mnttab(5).

NAME

diction - print wordy sentences
explain - interactive thesaurus for diction

SYNOPSIS

/usr/plx/diction [**-ml**] [**-mm**] [**-n**] [**-f** pfile] file ...
/usr/plx/explain

DESCRIPTION

Diction finds all sentences in a document that contain phrases from a data base of bad or wordy diction. Each phrase is bracketed with [ ]. Because *diction* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **-ms** may be overridden with the flag **-mm.** The flag **-ml** which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The user may supply her/his own pattern file to be used in addition to the default file with **-f** *pfile.* If the flag **-n** is also supplied the default file will be suppressed.

Explain is an interactive thesaurus for the phrases found by diction.

SEE ALSO

deroff(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

DIFF(1)

NAME

diff - differential file comparator

SYNOPSIS

diff [**-efbh**] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is **-**, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. 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 program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **-h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **-e** and **-f** are unavailable with **-h**.

FILES

/tmp/d?????
/usr/lib/diffh for **-h**

SEE ALSO

cmp(1), comm(1), ed(1), bdiff(1), diff3(1), sdiff(1), diffmk(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the **-e** or **-f** option are naive about creating lines consisting of a single period (.).

DIFF3(1)

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

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 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. Option **-x** (**-3**) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed - file1
```

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat **-e**.
Files longer than 64K bytes won't work.

NAME

diffmk - mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff*(1) or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, he can use *diffmk* to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file **macs** contains:

```
.pl 1
.ll 77
.nf
.eo
.nc \
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

SEE ALSO

diff(1), nroff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

NAME

dircmp - directory comparison

SYNOPSIS

dircmp [**-d**] [**-s**] dir1 dir2

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various 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.

The **-d** option causes *dircmp* to do a **diff**(1) of all the files of the same file name that are found to be different.

The **-s** option reports only files that are unique to one of the other directories, and files that have the same file names but are different. It does not report on files that are the same, or on directories.

SEE ALSO

cmp(1), diff(1).

NAME

dnld - download program files

SYNOPSIS

/etc/dnld [options]

DESCRIPTION

This program transfers program files from the UNIX system to either the EH 4A/BPS4 prom programmer or a DATA I/O prom programmer or a Plexus system that is running a debugging program. The program options are as follows:

-a *xxxx* Sets xxxx as the base address for text relocation. xxxx is a hex number. This address is also sent to the Plexus monitor if the program is in that mode.

-b *xxxx* Sets xxxx as the base address for bss relocation. xxxx is a hex number. This address is also sent to the Plexus monitor if the program is in that mode.

-i Initializes the EH-4A PROM programmer, does the *dnld*, and programs the PROM.

-c Puts a checksum (so that the words will sum to 0) at location 0x0ffe. Used for making PROMs so that they can be checked for integrity.

-t *info* If the output file is a tty then *info* is used to set up the terminals options. This is done by first opening the terminal and then issuing an *stty* command to it with *info* as the parameters.

-o *outf* Sets the output file name to *outf*.

-f *inf* Sets the input file name to *inf*.

-k *promsize* Determines the size of the proms being programmed.

-l Causes the low byte of each instruction in *inf* to be output to *outf*. Used only for prom programming.

-h Causes the high byte of each instruction in *inf* to be output to *outf*. Used only for prom programming.

-p Sets the program to output data in the format used by the EH prom programmer.

-z Sets the program to output data in the format used by the Plexus monitor.

-s *xxxx* Sets the segment number sent to the Plexus monitor. xxxx is a hex number.

-u Used for downloading UNIX thru the boot program,

-v Used for the 2732As.

-y *xxxx* Sets the communications address for loading the SIOC.

-d Used for downloading the ICP.

-B Used for 4B/BPS4 PROM programmer.

-D Used for the DATA I/O 29A programmer.

-F *xxyy* Used only for the DATA I/O programmer and must be present if the -D switch is. *xx* is the family and *yy* is the pinout code (e.g. 1924 for 2732DC).

-L Object file header contains LONGs as in 68000 type object files.

The default options are:

```
-a 0000
-b 0000
-t 1200
-o /dev/promio
-f a.out
-l
-p
-s 0000
-y f800
```

FILES

/dev/promio

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

icpdmp(1M)

BUGS

Some of the options may not work for programming proms.

NAME

du - summarize disk usage

SYNOPSIS

du [**-ars**] [names]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, . is used. Blocks are 1024 bytes long.

The optional argument **-s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **-a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

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 counted only once.

NOTES

Plexus provides a standalone version of *du* in addition to the one that runs under Sys3.

BUGS

If the **-a** option is not used, non-directories 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.

NAME

dump - incremental file system dump

SYNOPSIS

dump [key [arguments] file-system]

DESCRIPTION

Dump copies to magnetic tape all files changed after a certain date in the *file-system*. The *key* specifies the date and other options about the dump. *Key* consists of characters from the set **0123456789fusd**.

- **f** Place the dump on the next *argument* file instead of the tape.
- **u** If the dump completes successfully, write the date of the beginning of the dump on file **/etc/ddate**. This file records a separate date for each file system and each dump level.
- **0-9** This number is the "dump level". All files modified since the last date stored in the file **/etc/ddate** for the same file system at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire file system to be dumped.
- **s** The size of the dump tape is specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, the dump will wait for reels to be changed. The default size is 2,300 feet.
- **d** The density of the tape, expressed in BPI, is taken from the next *argument*. This is used in calculating the amount of tape used per write. The default is 1600.

If no arguments are given, the *key* is assumed to be **9u** and a default file system is dumped to the default tape.

Now a short suggestion on how to perform dumps. Start with a full level-0 dump: **dump 0u**. Next, periodic level-9 dumps should be made on an exponential progression of tapes. (Sometimes called Tower of Hanoi: 1, 2, 1, 3, 1, 2, 1, 4, ...; tape 1 used every other time, tape 2 is used every fourth, tape 3 is used every eighth, etc.): **dump 9u**. When the level-9 incremental approaches a full tape (about 78,000 blocks at 1600 BPI blocked 10 1024-byte blocks per record), a level-1 dump should be made: **dump 1u**. After this, the exponential series should progress as if uninterrupted. These level-9 dumps are based on the level-1 dump, which is based on the level-0 full dump. This progression of levels of dumps can be carried as far as desired.

FILES

default file system and tape vary with installation.
/etc/ddate: record dump dates of file system/level.

SEE ALSO

cpio(1), dumpdir(1M), restor(1M), volcopy(1M), dump(5).

DIAGNOSTICS

If the dump requires more than one tape, it will ask you to change tapes. Reply with a new-line after this has been done.

BUGS

Sizes are based on 1600 BPI blocked tape. The raw magnetic tape device has to be used to approach these densities. Read errors on the file system are ignored. Write errors on the magnetic tape are usually fatal.

NAME

dumpdir – print the names of files on a dump tape

SYNOPSIS

dumpdir [**f** filename]

DESCRIPTION

Dumpdir is used to read magtapes dumped with the *dump* command and list the names and inode numbers of all the files and directories on the tape.

The **f** option makes *filename* the name of the tape instead of the default.

FILES

Default tape unit varies with installation.
rst*

SEE ALSO

dump(1), restor(1)

DIAGNOSTICS

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

BUGS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, *dumpdir* doesn't use it.

Dumpdir cannot report correctly on a file having a very long directory path (greater than 15 directories).

NAME

echo - echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It 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
****	backslash
*****n*	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

NAME

ed - text editor

SYNOPSIS

ed [-] [**-x**] [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. The optional - suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*. If **-x** is present, an *x* command is simulated first to handle an encrypted file. *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.

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 typing 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 (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character REs* match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (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. $ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, **[0-9]** is equivalent to

[0123456789]. The - loses this special meaning if it occurs first (after an initial ^, 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 ^, if any); e.g., **[]a-f]** matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by **\{***m***\}**, **\{***m***,\}**, or **\{***m***,***n***\}** is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; **\{***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 RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences **\(** and **\)** is a RE that matches whatever the unadorned RE matches.

2.6 The expression *****n* matches the same string of characters as was matched by an expression enclosed between **\(** and **\)** *earlier* in the same RE. Here *n* is a digit; the sub-expression 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 RE* may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A currency symbol (**$**) at the end of an entire RE constrains that RE to match a *final* segment of a line. The construction **^***entire RE***$** constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. 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 lower-case letter. Lines are marked with the *k* command described below.

ED(1)

5. A RE 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 RE. 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. See also the last paragraph before *FILES* below.

6. A RE 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 RE. 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 (respectively 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 one(s) 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; they show that the given addresses are the default.

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; . is left at the last inserted line, or, if there were none, 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; . is left at 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; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* begins with **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file*

The *E*dit command is like *e*, except that 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*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE*/*command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. 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 multi-line 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 *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE*/

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . 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 new-line acts as a null command; an **&** 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 an interrupt signal (ASCII DEL or BREAK).

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 **?** 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; **.** is left at the last inserted line, or, if there were none, 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 new-line 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 lower-case letter. The address *'x* then addresses this line; **.** is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other 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; **.** 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; **.** is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; **.** is left at the last line printed. The *p* command may be appended to any other 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 (but see *DIAGNOSTICS* below).

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 file name is given, the currently-remembered file name, if any, is used (see *e* and *f*

commands). The currently-remembered file name is *not* changed unless *file* is the very first file name 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; . is set to the last line read in. If *file* begins with !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/**g**

The substitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *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 new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by ****. As a more general feature, the characters *****n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE 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 the *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 is preceded by a ****.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by ****. Such 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); . is left at 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/*RE*/*command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE*/

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 RE.

(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 writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file

name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); **.** is unchanged. If the command is successful, the number of characters written is typed. If *file* begins with **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name.

x

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)=

The line number of the addressed line is typed; **.** is unchanged by this command.

!*shell command*

The remainder of the line after the **!** is sent to the UNIX shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; 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; **.** is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. 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 RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. 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.

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file.
	(use the *h*elp command 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: it prints **?** and allows one to continue editing. A second *e* or *q* command at this point will take effect. The - command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1).
A Tutorial Introduction to the UNIX Text Editor by B. W. Kernighan.
Advanced Editing on UNIX by B. W. Kernighan.

CAVEATS AND BUGS

A *!* 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*(1)).
The sequence **\n** in a RE does not match any character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.
Because 0 is an illegal address for the *w* command, it is not possible to create an empty file with *ed*.

NAME

edit - text editor (variant of the ex editor for new or casual users)

SYNOPSIS

/usr/plx/edit [**-r**] name ...

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command oriented editor. The following brief introduction should help you get started with *edit.* A more complete basic introduction is provided by *Edit: A tutorial* . A *Ex/edit command summary (version 2.0)* is also very useful. See *ex*(1) for other useful documents; in particular, if you are using a CRT terminal you will want to learn about the display editor *vi.*

BRIEF INTRODUCTION

To edit the contents of an existing file you begin with the command "edit name" to the shell. *Edit* makes a copy of the file, and tells you how many lines and characters are in the file; you can then edit it. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but don't worry.

Edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* "buffer" (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you don't tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file, or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, and line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say "s/*old*/*new*/" where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but don't change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No **write** since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e. "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if give a **substitute** command which doesn't do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down, D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "''". The **z** command can also be given other following characters "z-" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines do, e.g., "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form /**text**/ to search forward for *text* or **?text?** to search backward for *text.* If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form /^**text**/ which searches for *text* at the beginning of a line. Similarly /**text$**/ searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$-5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20move "a" which deletes these lines from the file and places them in a buffer named *a. Edit* has 26 such buffers named *a* through *z.* You can later get these lines back by doing ""a move ." to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines, following it with the name of the other file you wish to edit, i.e. "edit chapter2". By changing *move* to *copy* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

NOTES

This command is based on one developed at the University of California at Berkeley.

SEE ALSO

ex (1), vi (1), 'Edit: A tutorial', by Ricki Blau and James Joyce

BUGS

See *ex*(1).

NAME

efl - Extended Fortran Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the EFL language into clean Fortran on the standard output. *Efl* provides the C-like control constructs of *ratfor*(1):

statement grouping with braces.

decision-making:

if, **if-else**, and **select-case** (also known as **switch-case**);
while, **for**, Fortran **do**, **repeat**, and **repeat** ... **until** loops;
multi-level **break** and **next**.

EFL has C-like data structures, e.g.:

```
struct
        {
        integer flags(3)
        character(8) name
        long real coords(2)
        } table(100)
```

The language offers generic functions, assignment operators (+=, &=, etc.), and sequentially evaluated logical operators (&& and ||). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) } )
```

EFL also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:
this is a comment.

translation of relational and logical operators:
>, >=, &, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:
return (*expression***)**

defines:
define *name replacement*

includes:
include *file*

Efl understands several option arguments: **-w** suppresses warning messages, **-#** suppresses comments in the generated program, and the default option **-C** causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on. Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per

word, and default formats.

Efl is best used with *f77*(1), Plexus product number 4214A (for P/35 and P/60) and 4108A (for P/25 and P/40).

SEE ALSO

cc(1), ratfor(1).
The Programming Language EFL by S.I. Feldman.

NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable printers
disable [**–c**] [**–r**[reason]] printers

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

–c Cancel any requests that are currently printing on any of the designated printers.

–r[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned 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 will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

lp(1), lpstat(1).

NAME

env - set environment for command execution

SYNOPSIS

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 resulting environment is printed, one name-value pair per line.

Variables can be added to the environment by the *export* command inherent within shell.

SEE ALSO

sh(1), exec(2), profile(5), environ(7).

NAME

eqn, neqn, checkeq – format mathematical text for nroff or troff

SYNOPSIS

eqn [**–d**xy] [**–p**n] [**–s**n] [**–f**n] [files]

neqn [**–d**xy] [**–p**n] [**–s**n] [**–f**n] [files]

checkeq [files]

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematical text on a Wang Laboratories, Inc. C/A/T phototypesetter, while *neqn* is used for the same purpose with *nroff*(1) on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent.

If no files are specified, these programs read from the standard input. A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN.** Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **–d***xy* or (more commonly) with **delim** *xy* between **.EQ** and **.EN.** The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and **.EQ**/**.EN** pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub j* makes x_j, *a sub k sup 2* produces a_k^2, while $e^{x^2+y^2}$ is made with *e sup {x sup 2 + y sup 2}*. Fractions are made with **over**: *a over b* yields $\frac{a}{b}$; **sqrt** makes square roots: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits: $\lim_{n\to\infty}\sum_{0}^{n} x_i$ is made with *lim from {n –> inf } sum from 0 to n x sub i*. Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~=~ 1* produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1$. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket). A **left** *thing* need not have a matching **right** *thing*.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies. Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition,

there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~=~ n under* is $\ddot{\bar{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \vec{y}$.

Point sizes and fonts can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **–s***n* and **–f***n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **–p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define thing % replacement %

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** (∑), **int** (∫), **inf** (∞), and shorthands such as >= (≥), != ( ), and –> (→) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or **GAMMA** (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as **\(dd** (‡) and **\(bs** (\) may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff*(1) when all else fails. Full details are given in the manual cited below.

SEE ALSO

Typesetting Mathematics–User's Guide by B. W. Kernighan and L. L. Cherry.
New Graphic Symbols for EQN and NEQN by C. Scrocca.
mm(1), mmt(1), tbl(1), troff (1), eqnchar(7), mm(7), mv(7).

BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.
See also *BUGS* under *troff*(1).

NAME

errdead - extract error records from dump

SYNOPSIS

/etc/errdead dumpfile [namelist]

DESCRIPTION

When hardware errors are detected by the system, an error record that contains information pertinent to the error is generated. If the error-logging daemon *errdemon*(1M) is not active or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. *Errdead* examines a system dump (or memory), extracts such error records, and passes them to *errpt*(1M) for analysis.

The *dumpfile* specifies the file (or memory) that is to be examined. The system namelist is specified by *namelist*; if not given, /**sys3** is used.

FILES

/sys3	system namelist
/usr/bin/errpt	analysis program
/usr/tmp/errXXXXXX	temporary file

DIAGNOSTICS

Diagnostics may come from either *errdead* or *errpt*. In either case, they are intended to be self-explanatory.

SEE ALSO

errdemon(1M), errpt(1M).

NAME

errdemon - error-logging daemon

SYNOPSIS

/usr/lib/errdemon [file]

DESCRIPTION

The error logging daemon *errdemon* collects error records from the operating system by reading the special file **/dev/error** and places them in *file*. If *file* is not specified when the daemon is activated, **/usr/adm/errfile** is used. Note that *file* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon*; that responsibility is left to *errpt*(1M). The error-logging daemon is terminated by sending it a software kill signal (see *signal*(2)). Only the super-user may start the daemon, and only one daemon may be active at any time.

FILES

/usr/lib/error	source of error records
/usr/adm/errfile	repository for error records

DIAGNOSTICS

The diagnostics produced by *errdemon* are intended to be self-explanatory.

SEE ALSO

errpt(1M), errstop(1M), kill(1), err(4).

NAME

errpt - process a report of logged errors

SYNOPSIS

errpt [**-a**] [-dev]... [**-int**] [**-mem**] [**-s** date] [**-e** date] [**-pn**] [**-f**] [files]

DESCRIPTION

Errpt processes data collected by the error logging mechanism (*errdemon*(1M)) and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, *errpt* attempts to use **/usr/adm/errfile** as *file*.

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unabled to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times that *errpt* has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped, and any time changes (via *date*(1)) that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report may be limited to certain records in the following ways:

-s *date*	Ignore all records posted earlier than *date*, where *date* has the form **mmddhhmmyy**, consistent in meaning with the *date*(1) command.
-e *date*	Ignore all records posted later than *date*, whose form is as described above.
-a	Produce a detailed report that includes all error types.
-dev	A detailed report is limited to *dev*, a block device identifier. *Errpt* is familiar with the common form of identifiers (e.g., PD, is; see Section 4 of this volume). Currently, the block devices for which errors are logged are PD, PT, IS, and RM.
-int	Include in a detailed report errors of the stray-interrupt type.
-mem	Include in a detailed report errors of the memory-parity type.
-p *n*	Limit the size of a detailed report to *n* pages.
-f	In a detailed report, limit the reporting of block device errors to unrecovered errors.

FILES

/usr/adm/errfile default error file

SEE ALSO

errdemon(1M), errfile(5).

BUGS

The information about block devices is incomplete, and may be incorrect.

NAME

errstop - terminate the error-logging daemon

SYNOPSIS

/etc/errstop [namelist]

DESCRIPTION

The error-logging daemon *errdemon*(1M) is terminated by using *errstop*. This is accomplished by executing *ps*(1) to determine the daemon's identity and then sending it a software kill signal (see *signal*(2)); **/sys3** is used as the system namelist if none is specified. Only the super-user may use *errstop*.

FILES

/sys3 default system namelist

DIAGNOSTICS

The diagnostics produced by *errstop* are intended to be self-explanatory.

SEE ALSO

errdemon(1M), ps(1), kill(2).

NAME

ex - text editor

SYNOPSIS

/usr/plx/ex [**-**] [**-v**] [**-t** tag] [**-r**] [+*lineno*] name ...

DESCRIPTION

Ex is the root of a family of editors: *edit, 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.*

If you have not used *ed,* or are a casual user, you may find that the editor *edit* is most convenient for you. It avoids some of the complexities of *ex* used mostly by systems programmers and persons very familiar with *ed.*

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex.*

DOCUMENTATION

For *edit* and *ex* see the *Ex/edit command summary - Version 2.0.* The document *Edit: A tutorial* provides a comprehensive introduction to *edit* assuming no previous knowledge of computers or the UNIX system.

The *Ex Reference Manual - Version 2.0* is a comprehensive and complete manual for the command mode features of *ex,* but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex*, see the editing documents written by Brian Kernighan for the editor *ed*; the material in the introductory and advanced documents works also with *ex.*

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi.* The *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction.*

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi.* Generally, the editor uses far more capabilities of terminals than *ed* does. It also uses the terminal capability data base *termcap*(1) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1). There is also an interline editing **open** (**o**) command which works on all terminals.

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you don't accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, filenames in the editor may be formed with full shell

metasyntax. The metacharacter '%' is also available in forming filenames and is replaced by the name of the current file. For editing large groups of related files you can use *ex's* **tag** command to quickly locate functions and other important points in any of the files. This is useful when working on a large program when you want to quickly find the definition of a particular function. The command *ctags*(1) builds a *tags* file or a group of C programs.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

You can use the **substitute** command in *ex* to systematically convert the case of letters between upper and lower case. It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join** (j) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort.*

FILES

/usr/lib/ex2.0strings	error messages
/usr/lib/ex2.0recover	recover command
/usr/lib/ex2.0preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

awk(1), ed(1), grep(1), sed(1), edit(1), grep(1), termcap(5), vi(1)

BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line '-' option is used.

There is no easy way to do a single scan ignoring case.

Because of the implementation of the arguments to *next,* only 512 bytes of argument list are allowed there.

The format of */etc/termcap* and the large number of capabilities of terminals used by the editor cause terminal type setup to be rather slow.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

NAME

expr - evaluate arguments as an expression

SYNOPSIS

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 **0** 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. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

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 or **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.

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*(1), except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** 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 equal to either "/usr/abc/file" or just "file" '
   expr $a : '.*/\(.*\)' \| $a

   returns the last segment of a path name (i.e., file). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. # A better representation of example 2.
   expr //$a : '.*/\(.*\)'

   The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. expr $VAR : '.*'

   returns the number of characters in **$VAR**.

SEE ALSO

ed(1), sh(1).

EXIT CODE

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

DIAGNOSTICS

syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string

BUGS

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 =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

NAME

file - determine file type

SYNOPSIS

file [**-f**] file ...

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 an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0 (see the description of the **-V** option in *ld*(1)).

If the **-f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

NAME

find - find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below. 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 **[**, **?** and ***).

-perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

(flags&onum)===onum

-type *c* True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **f**, or **p** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file.

-links *n* True if the file has *n* links.

-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, it is taken as a group ID.

-size *n* True if the file is *n* blocks long (1024 bytes per block).

-atime *n* True if the file has been accessed in *n* days.

-mtime *n* True if the file has been modified in *n* days.

-ctime *n* True if the inode of the file has been modified in *n* days. Inodes are changed by *chmod*, *chown*, *chgrp*, *ln*, *rm*, and by changing the length of the file.

-exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current path name.

-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**.

-print Always true; causes the current path name to be printed.

-cpio *device* Write the current file on *device* in *cpio* (5) format (5120 byte records).

-newer *file* True if the current file has been modified more recently than the argument *file*.

-inum *n* True if the inode number of the current path name is *n*.

(*expression* **)** True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (**!** is the unary *not* operator).
2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**-o** is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

FILES

/etc/passwd, /etc/group

SEE ALSO

cpio(1), sh(1), test(1), stat(2), cpio(5), fs(5).

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [**-y**] [**-n**] [**-sX**] [**-SX**] [**-t** file] [file-system] ...

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for UNIX file systems. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **-n** action.

Fsck has more consistency checks than its predecessors *check*, *dcheck*, *fcheck*, and *icheck* combined.

The following flags are interpreted by *fsck*.

-y Assume a yes response to all questions asked by *fsck*.

-n Assume a no response to all questions asked by *fsck*; do not open the file system for writing.

-s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The **-s***X* option allows for creating an optimal free-list organization. The following form of *X* is supported:

-sBlocks-per-cylinder:Blocks-to-skip (for anything else)

If *X* is not given, the values used when the file system was created are used. If these values were not specified, then the value *400:9* is used.

-S*X* Conditionally reconstruct the free list. This option is like **-s***X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **-S** will force a no response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

-t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, *fsck* will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   - Incorrect number of blocks.
   - Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   - File pointing to unallocated inode.
   - Inode number out of range.
8. Super Block checks:
   - More than 65536 inodes.
   - More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster.

FILES

/etc/checklist contains default list of file systems to check.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

Fsck can generate the message "possible file size error" with an inode number. This message may be given when *fsck* encounters a file containing a large group of null characters between areas of text (the file has "holes" in it). UNIX does not allocate disk blocks for the nulls, so the size of the file as determined by *fsck* looks smaller than it should be. To figure out if you really have a problem, run *ncheck* with the **-i** option to determine which file is at fault. Then check the file; if it contains lots of nulls together, the file system is probably okay.

NOTES

In addition to the stock *fsck*, Plexus provides a standalone version.

SEE ALSO

checklist(5), fs(5), crash(8).
FSCK - The UNIX/TS File System Check Program, by T. J. Kowalski.

BUGS

Inode numbers for . and .. in each directory should be checked for validity.

-g and **-b** options from *check* should be available in *fsck*.

Fsck does not handle fifos properly.

Fsck prints a message to reboot the system with no *sync* when it corrects the root file system. Rebooting too quickly after that message invalidates the corrections.

The standalone version of *fsck* can handle at most five file systems.

NAME

fsdb - file system debugger

SYNOPSIS

/etc/fsdb special [-]

DESCRIPTION

Fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional - argument or by the use of the **O** symbol. (*Fsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,-	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. .The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words

e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
s0	high byte of file size
s1	low word of file size
a#	data block numbers (0 - 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.
2i.fd	prints the first 64 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first 1024 bytes of the file are then printed in ASCII.
1b.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.

NOTES

Plexus provides a standalone version of *fsdb* in addition to the one that runs under Sys3.

SEE ALSO

fsck(1M), dir(5), fs(5).

NAME

fwtmp, wtmpfix - manipulate wtmp records

SYNOPSIS

/usr/lib/acct/fwtmp [-ic]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

Fwtmp

Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formated ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument **-ic** is used to denote that input is in ASCII form, and output is to be written in binary form.

Wtmpfix

Wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of *files* to indicate the standard input. If time/date corrections are not made, *acctcon1* will fault when it encounters certain date change records.

Each time the date is set while operating in multi-user mode, a pair of date change records are written to **/usr/adm/wtmp**. The first record is the old date denoted by | in the name field. The second record specifies the new date and is denoted by a { in the name field. *Wtmpfix* uses these records to synchronize all time stamps in the file.

FILES

/usr/adm/wtmp
/usr/include/utmp.h

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M), acct(2), acct(5), utmp(5).

NAME

get - get a version of an SCCS file

SYNOPSIS

get [**-r**SID] [**-c**cutoff] [**-i**list] [**-x**list] [**-a**seq-no.] [**-k**] [**-e**] [**-l**[p]] [**-p**] [**-m**] [**-n**] [**-s**] [**-b**] [**-g**] [**-t**] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with **-**. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the **-e** keyletter is also used), as a function of the SID specified.

-c*cutoff* *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **-c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: **"-c77/2/2 9:22:25"**. Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

~!get "-c%E% %U%" s.file

-e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The **-e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get -e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **-e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **-k** keyletter in place of the **-e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **-e** keyletter is used.

-b Used with the **-e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

-i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

-x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **-i** keyletter for the *list* format.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** keyletter is implied by the **-e** keyletter.

-l[p] Causes a delta summary to be written into an *l-file*. If **-lp** is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **-s** keyletter is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **-m** keyletter generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

-t Used to access the most recently created ("top") delta in a given release (e.g., **-r1**), or release and level (e.g., **-r1.2**).

-a*seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the **-r** and **-a** keyletters are specified, the **-a** keyletter is used. Care should be taken when using the **-a** keyletter in conjunction with the **-e** keyletter, as the SID of the delta to be created may not be what one expects. The **-r** keyletter can be used with the **-a** and **-e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** keyletter is used included deltas are listed following the notation "Included"; if

the **-x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**-b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	-	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

\# Successor.

† The **-b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of - means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.

%L% Level.
%B% Branch.
%S% Sequence.
%D% Current date (YY/MM/DD).
%H% Current date (MM/DD/YY).
%T% Current time (HH:MM:SS).
%E% Date newest applied delta was created (YY/MM/DD).
%G% Date newest applied delta was created (MM/DD/YY).
%U% Time newest applied delta was created (HH:MM:SS).
%Y% Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F% SCCS file name.
%P% Fully qualified SCCS file name.
%Q% The value of the **q** flag in the file (see *admin*(1)).
%C% Current line number. This keyword is intended for identifying messages output by the program such as "this shouldn't have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z% The 4-character string **@(#)** recognizable by *what*(1).
%W% A shorthand notation for constructing *what*(1) strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
%A% Another shorthand notation for constructing *what*(1) strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*, These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **-p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **-k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **-l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   * if the delta wasn't applied and wasn't ignored.
c. A code indicating a "special" reason why the delta was or was not applied:
   "I": Included.
   "X": Excluded.
   "C": Cut off (by a **-c** keyletter).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.

i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an **-e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **-e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **-i** keyletter argument if it was present, followed by a blank and the **-x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the **-e** keyletter is used.

NAME

getopt - parse command options

SYNOPSIS

set -- \` getopt optstring $* \`

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures, and to check for legal options. *Optstring* is a string of recognized option letters (see getopt(3C)); 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 white space. 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 - and in its own shell argument; each option argument is also in its own shell argument.

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

EXAMPLE

The following code fragment shows how one might 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
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b) FLAG=$i; shift;;
        -o)             OARG=$2;    shift; shift;;
        --)             shift;   break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

sh(1), getopt(3C).

NAME

graph - draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *tplot*(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ", in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **-x**).

-b Break (disconnect) the graph after each label in the input.

-c Character string given by next argument is default label for each point.

-g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

-l Next argument is label for graph.

-m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

-s Save screen, don't erase before plotting.

-x [l] If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

-y [l] Similarly for *y*.

-h Next argument is fraction of space for height.

-w Similarly for width.

-r Next argument is fraction of space to move right before plotting.

-u Similarly to move up before plotting.

-t Transpose horizontal and vertical axes. (Option **-x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **-s** option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

graphics(1G), spline(1G), tplot(1G).

BUGS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

greek - select terminal filter

SYNOPSIS

greek [**-T**terminal]

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 (which is the *nroff*(1) default 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*(7)). The following *terminal*s are recognized currently:

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(1), 300s(1), 4014(1), 450(1), eqn(1), greek(7), hp(1), mm(1), nroff(1), tplot(1G), environ(7), term(7).

NAME

grep, egrep, fgrep - search a file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output.

The three flavors of *grep* find different kinds of patterns. *Grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. *Grep* cannot find patterns in **a.out** files, though *egrep* and *fgrep* can.

The following *options* are recognized:

-v All lines but those matching are printed.
-x (Exact) only lines matched in their entirety are printed (*fgrep* only).
-c Only a count of matching lines is printed.
-l Only the names of files with matching lines are listed (once), separated by new-lines.
-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 The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).
-e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a - (does not work with *grep*).
-f *file* The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name 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* argument in single quotes **'...'**.

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

Egrep accepts regular expressions as in *ed*(1), except for **\(** and **\)**, with the addition of:

1. A regular expression followed by **+** matches one or more occurrences of the regular expression.
2. A regular expression followed by **?** matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by **|** or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses **()** for grouping.

The order of precedence of operators is **[]**, then *** ? +**, then concatenation, then **|** and new-line.

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

BUGS

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters; longer lines are truncated.
Egrep does not recognize ranges, such as **[a-z]**, in character classes.

NAME

head - give first few lines of a stream

SYNOPSIS

/usr/plx/head [-count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

tail(1)

NAME

help - ask for help

SYNOPSIS

help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **ge6**, for message 6 from the *get* command).

type 2 Does not contain numerics (as a command, such as **get**)

type 3 Is all numeric (e.g., **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

FILES

/usr/lib/help directory containing files of message text.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

hp - handle special functions of HP 2640 and 2621-series terminals

SYNOPSIS

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*(1) output. Typical uses are:

```
nroff -h files ... | hp
nroff -h -s ... files | hp
```

In the latter case, *nroff* will stop at the beginning of each page (including the first) and wait for you to hit line-feed (control-j) before resuming output.

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 Underline 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 removal of new-lines. 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 does *300*(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*(1).

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(1), col(1), greek(1), neqn(1), tbl(1), troff(1).

BUGS

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*(1) 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*(1).
Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hyphen - find hyphenated words

SYNOPSIS

hyphen files

DESCRIPTION

Hyphen finds all the hyphenated words in *files* and prints them on the standard output. If no arguments are given, the standard input is used. Thus *hyphen* may be used as a filter.

BUGS

Hyphen can't cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

icpdmp - take a core image of the ICP and transfer to a host file

SYNOPSIS

/etc/icpdmp device filename

DESCRIPTION

Icpdmp takes a core image of the Intelligent Communications Processor (ICP) and transfers it to a host file. *Device* is the download device for the ICP, **/dev/ic[0-3]**.

This command is accessible only to the system administrator and has an important side effect: it does not leave the ICP intact. You must download, (using *dnld*(1M)), the ICP again before the ICP is usable. Normally, this means rebooting the system.

This command dumps all of the ICP RAM (4000 hex to BFFF hex -- 32K-bytes).

This command is rarely useful except if the ICP crashes hard.

FILES

/dev/ic[0-3]

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

dnld(1M), tty(4)

ID(1)

NAME

id - print user and group IDs and names

SYNOPSIS

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(1), getuid(2), getgid(2).

NAME

install - install commands

SYNOPSIS

install [**-c** dira] [**-f** dirb] [**-i**] [**-n** dirc] [**-o**] [**-s**] file [dirx ...]

DESCRIPTION

Install is a command most commonly used in "makefiles" (see *make*(1)) to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, *install* will search (using *find*(1)) a set of default directories (**/bin**, **/usr/bin**, **/etc**, **/lib**, and **/usr/lib**, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

-c *dira* — Installs a new command in the directory specified in *dira*. Looks for *file* in *dira* and installs it there if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **-s** option.

-f *dirb* — Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to **755** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **-o** or **-s** options.

-i — Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options other than **-c** and **-f**.

-n *dirc* — If *file* is not found in any of the searched directories, it it put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options other than **-c** and **-f**.

-o — If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. May be used alone or with any other options other than **-c**.

-s — Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

mk(8).

NAME

join - relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is -, 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 new-line. 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, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

-e *s* Replace empty output fields by string *s*.

-j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file.

-o *list* Each output line comprises the fields specifed 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* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

SEE ALSO

awk(1), comm(1), sort(1).

BUGS

With default field separation, the collating sequence is that of **sort -b**; with **-t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk*(1) are wildly incongruous.

NAME

kill - terminate a process

SYNOPSIS

kill [-signo] processid ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous 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*(1).

The details of the kill are described in *kill*(2). 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 first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill -9 . . ." is a sure kill.

SEE ALSO

ps(1), sh(1), kill(2), signal(2).

NAME

ld - link editor

SYNOPSIS

ld [**-sulxXrdnim**] [**-o** name] [**-t** name] [**-V** num] file ...

DESCRIPTION

Ld combines several object programs into one; resolves external references; and searches libraries (as created by *ar*(1)). In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the **-r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out**. This file is made executable if no errors occurred during the load and the **-r** flag was not specified.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

The loader accepts 8-character symbols, and the first character of each symbol is an underbar ('_'), which *cc* prefixes at compile time. Therefore, symbol names in program modules that are to be linked must be unique within the first seven characters.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

The symbols **_etext**, **_edata** and **_end** (**etext**, **edata** and **end** in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several flag arguments which are written preceded by a **-**. Except for **-l**, they should appear before the file names.

-s "Strip" the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1). This option is turned off if there are any undefined symbols.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-l This option is an abbreviation for a library name. **-l** alone stands for **/lib/libc.a**, which is the standard system library for C and assembly language programs. **-l***x* stands for **/lib/lib***x***.a**, where *x* is a string. If that does not exist, *ld* tries **/usr/lib/lib***x***.a** A library is searched when its name is encountered, so the placement of a **-l** is significant.

-x Do not preserve local (non-**.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

-X Save local symbols except for those whose names begin with **L**. This option is used by *cc* to discard internally generated labels while retaining symbols local to routines.

-r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the "undefined symbol" diagnostics.

-d Force definition of common storage even if the **-r** flag is present.

-n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 2K-byte (Z8000) or 4K-byte (MC68000) boundary following the end of

the text. On the MC68000, this option is on by default; use **-N** to turn it off.

-i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and **-n** is that here the data starts at location 0. This option is meaningful only on the Z8000; it does nothing on the MC68000.

-m The names of all files and archive members used to create the output file are written to the standard output. (Z8000 only)

-o The *name* argument after **-o** is used as the name of the *ld* output file, instead of **a.out**.

-t The *name* argument is taken to be a symbol name, and any references to or definitions of that symbol are listed, along with their types. There can be up to 16 occurrences of **-t***name* on the command line. (Z8000 only)

-V The *num* argument is taken as a decimal version number identifying the **a.out** that is produced. *Num* must be in the range 0-32767. The version stamp is stored in the **a.out** header; see *a.out*(5). (Z8000 only)

FILES

/lib/lib?.a	libraries
/usr/lib/lib?.a	more libraries
a.out	output file

SEE ALSO

ar(1), as(1), cc(1), a.out(5).

NAME

lex - generate programs for simple lexical tasks

SYNOPSIS

lex [**-rctvn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found. The *EXAMPLE* below shows a typical *lex* program.

Running *lex* on *files* generates the file **lex.yy.c**, a C program. Compile **lex.yy.c** with the *-ll* option, which loads the *lex* library, as follows

cc lex.yy.c -ll

The **a.out** created by this step, when run with another program as its argument, copies the input to the output except when a string specified in the original *file* is found; then the corresponding program text is executed.

The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx-z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *, +, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character . is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation *r*{*d*,*e*} in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than |, but lower than *, ?, +, and concatenation. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character **$** at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \. Thus **[a-zA-Z]**+ matches a string of letters.

Three subroutines defined as macros are expected: **input()** to read a character; **unput(***c***)** to replace a character read; and **output(***c***)** to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same *yytext*; and the function **yyless(***p***)** pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext*+*yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes **%%** it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a **%%**, as in YACC. Lines preceding **%%** which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with **{}**. Note that curly brackets do not imply parentheses; only string substitution is done.

EXAMPLE

```
D       [0-9]
%%
if      printf("IF statement\n");
[a-z]+  printf("tag, value %s\n",yytext);
```

```
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

The external names generated by *lex* all begin with the prefix **yy** or **YY**.

The flags must appear before any files. The flag **-r** indicates RATFOR actions, **-c** indicates C actions and is the default, **-t** causes the **lex.yy.c** program to be written instead to standard output, **-v** provides a one-line summary of statistics of the machine generated, **-n** will not print out the - summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n* number of positions is *n* (default 2000)

%n *n* number of states is *n* (500)

%t *n* number of parse tree nodes is *n* (1000)

%a *n* number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **-v** option, unless the **-n** option is used.

FILES

/usr/lib/libl.a lex library

SEE ALSO

yacc(1).
LEX - Lexical Analyzer Generator by M. E. Lesk and E. Schmidt.

BUGS

The **-r** option is not yet fully operational.

NAME

line - read one line

SYNOPSIS

line

DESCRIPTION

Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1), read(2).

NAME

link, unlink - exercise link and unlink system calls

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

Link and *unlink* perform their respective system calls on their arguments, abandoning all error checking. These commands may only be executed by the super-user, who (it is hoped) knows what he or she is doing.

SEE ALSO

rm(1), link(2), unlink(2).

NAME

lint - a C program checker

SYNOPSIS

lint [**-abchnpuvx**] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things which are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

It is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. By default, *lint* uses function definitions from the standard lint library **llib-lc.ln**; function definitions from the portable lint library **llib-port.ln** are used when *lint* is invoked with the **-p** option.

Any number of *lint* options may be used, in any order. The following options are used to suppress certain kinds of complaints:

-a Suppress complaints about assignments of long values to variables that are not long.

-b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in a large number of such complaints.)

-c Suppress complaints about casts that have questionable portability.

-h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

-u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program.)

-v Suppress complaints about unused arguments in functions.

-x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint's* behavior:

-n Do not check compatibility against either the standard or the portable lint library.

-p Attempt to check portability to other dialects (IBM and GCOS) of C.

The **-D**, **-U**, and **-I** options of *cc*(1) are also recognized as separate arguments.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*VARARGS*n*/
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
turns on the **-v** option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

Lint produces its first output on a per source file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

FILES

/usr/lib/lint[12]	programs
/usr/lib/llib-lc.ln	declarations for standard functions (binary format; source is in **/usr/lib/llib-lc**)
/usr/lib/llib-port.ln	declarations for portable functions (binary format; source is in **/usr/lib/llib-port**)
/usr/tmp/*lint*	temporaries

SEE ALSO

cc(1).

BUGS

Exit(2) and other functions which do not return are not understood; this causes various lies.

NAME

login - sign on

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It can no longer be invoked explicitly, but is invoked by the system when a connection is first established, or after the previous user has logged out by sending an "end-of-file" (control-D) to his or her initial shell. (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

Login asks for your user name, and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "external" password. This will occur only for dial-up connections, and will be prompted by the message "External security:". Both passwords are required for a successful login.

If password aging has been invoked by the super-user on your behalf, your password may have expired. In this case, you will be shunted into *passwd*(1) to change it, after which you may attempt to login again.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, you will be informed of the existence (if any) of mail, and the profiles (i.e., **/etc/profile** and **$HOME/.profile**) (if any) are executed (see *profile*(5)). *Login* initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(1)) according to specifications found in the **/etc/passwd** file. Argument 0 of the command interpreter is - followed by the last component of the interpreter's path name. The *environment* (see *environ*(7)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
LOGNAME=*your-login-name*

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
$HOME/.profile	personal profile

SEE ALSO

mail(1), newgrp(1), sh(1), passwd(1), su(1), passwd(5), profile(5), environ(7), getty(8).

DIAGNOSTICS

Login incorrect
if the user name or the password is incorrect.

No shell, cannot open password file, no directory:
consult a UNIX programming counselor.

Your password has expired. Choose a new one.
if password aging is implemented.

NAME

logname - get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3X), environ(7).

NAME

lorder - find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

This brash one-liner intends to build a new library from existing **.o** files.

```
ar cr library `lorder *.o | tsort`
```

FILES

*symref, *symdef temp files

SEE ALSO

ar(1), ld(1), tsort(1).

BUGS

Object files whose name do not end with **.o**, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

lp [**–c**] [**–d**dest] [**–m**] [**–n**number] [**–o**option] [**–s**] [**–t**title] [**–w**] files
cancel [ids] [printers]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name – stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

–c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. 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* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). 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*(1)).

–m Send mail (see *mail(1)*) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

–n*number* Print *number* copies (default of 1) of the output.

–o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **–o** keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin*(1M).

–s Suppress messages from *lp*(1) such as "request id is ...".

–t*title* Print *title* on the banner page of the output.

–w Write 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.

Cancel cancels line printer requests that were made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which 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.

FILES

/usr/spool/lp/*

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

accept(1M), enable(1), lpadmin(1M), lpsched(1M), lpstat(1), mail(1).

NAME

lpadmin – configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin -pprinter [options]
/usr/lib/lpadmin -xdest
/usr/lib/lpadmin -d[dest]

DESCRIPTION

Lpadmin configures LP spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination. *Lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **-p**, **-d** or **-x** options must be present for every legal invocation of *lpadmin*.

-d[*dest*] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **-d**.

-x*dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **-x**.

-p*printer* names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.

The following *options* are only useful with **-p** and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

-c*class* inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.

-e*printer* copies an existing *printer's* interface program to be the new interface program for *P*.

-f indicates that *P* is set up for printing formfeeds on queue empty.

-h indicates that the device associated with *P* is hardwired. This *option* is assumed when creating a new printer unless the **-l** *option* is supplied.

-i*interface* establishes a new interface program for *P*. *Interface* is the pathname of the new program.

-l indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*(1M), disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

-m*model* selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP software (see *Models* below).

-n indicates that *P* is not set up for printing formfeeds on queue empty. This option is assumed when creating a new printer unless the "**-f**" option is supplied.

-r*class* removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.

-v*device* associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by the LP administrator, *lp*. Note that there is nothing to stop an administrator from associating the same *device* with more than one *printer*. If only the **-p** and **-v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the **–v** option and one of the **–e**, **–i** or **–m** options must be supplied. Only one of the **–e**, **–i** or **–m** options may be supplied. The **–h** and **–l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and _ (underscore).

Models.

Model printer interface programs are supplied with the LP software. They are shell procedures that interface between *lpsched (1M)* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **–m**. Models should have 644 permission if owned by lp & bin, or 664 permission if owned by bin & bin. Alternatively, LP administrators may modify copies of models and then use *lpadmin* **–i** to associate them with printers. The following list describes the *models* and lists the options which they may be given on the *lp* command line using the **–o** keyletter:

dumb interface for a line printer without special functions and protocol. Form feeds are assumed. This is a good model to copy and modify for printers which do not have models.

1640 DIABLO 1640 terminal running at 1200 baud, using XON/XOFF protocol. Options:

–12 12-pitch (10-pitch is the default)
–f do not use the *450*(1) filter. The output has been pre-processed by either *450*(1) or the *nroff (1)* 450 driving table.

interface a generalized interface program written in C which uses the *printcap*(5) database. It provides the ability to print formfeeds on queue empty (using *tr* in the *printcap* database for formfeed string), and handle special forms (using *sp* in *printcap* for the operator's device). See *printcap*(5) for information on other capabilities.

S[*string*]

use the special form whose name or identification number is *string*. If this option is used, the following interaction with the operator occurs when the request is to be printed. The **interface** program types out

MOUNT SPECIAL FORMS <string> on printer <printer name>

The program waits a few seconds. Then it types out

READY? Type "lpforms <printer name>"

When the operator types in "lpforms <printer name>", the program asks

Forms ready? or Hold ("Y" or "H")

If the operator types "Y", the request is printed. If the operator types "H", the request is put on hold; i.e., it remains in the print queue, awaiting an *lprun*(1) command to take it out of the hold state.

After the file has been printed, a similar interaction takes place to restore the standard forms.

hp Hewlett-Packard 2631A line printer at 2400 baud. Options:

–c compressed print
–e expanded print

prx Printronix P300 or P600 printer using XON/XOFF protocol at 1200 baud.

EXAMPLES

1. Assuming there is an existing Hewlett-Packard 2631A line printer named *hp2*, it will use the **hp** model interface after the command:

   /usr/lib/lpadmin –php2 –mhp

2. To obtain compressed print on *hp2*, use the command:

   lp –dhp2 –o–c files

3. A DIABLO 1640 printer called *st1* can be added to the LP configuration with the command:

   /usr/lib/lpadmin –pst1 –v/dev/tty20 –m1640

4. An *nroff (1)* document may be printed on *st1* in any of the following ways:

   nroff –T450 files | lp –dst1 –of
   nroff –T450–12 files | lp –dst1 –of
   nroff –T37 files | col | lp –dst1

5. The following command prints the password file on *st1* in 12-pitch:

   lp –dst1 –o12 /etc/passwd

   NOTE: the **–12** option to the **1640** model should never be used in conjunction with *nroff*(1).

FILES

/usr/spool/lp/*

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

EXIT CODES

Interface programs may pass the following exit codes to *lpsched*.

EXIT CODE	Meaning to lpsched
128	Must be set in order for 64 and 32 to be interpreted.
64	Printer ready for next request.
32	Sets current request as "not printing". Leaves it in queue instead of removing it. Resets printer.
16	Deletes request from queue and resets printer.
8	Sets flag that request is on hold -- stays in queue but is not printed. Sends message to **lpsched** that printer is ready for another request.
4	Removes request from queue and disables printer.
0	Removes request from the print queue, resets printer, and tells *lpsched* that printer is ready for the next request.

These requests are used bitwise, so only codes 3, 2, and 1 are left for user interface programs.

If *lpsched* receives a code that is not one of the above, the Plexus interface program **/usr/spool/lp/model/interface** does standard error handling. This consists of removing the current request from the print queue, resetting the printer, sending a message to the user, and telling *lpsched* that the printer is ready for the next request.

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1), nroff(1).

NAME

lpd - line printer daemon

SYNOPSIS

/usr/lib/lpd

DESCRIPTION

Lpd is the daemon for the line printer. It uses the directory **/usr/spool/lpd**. The file **lock** is used to prevent two daemons from becoming active. After the program has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with "df". Each such file is submitted as a job. Each line of a job file must begin with a key character to specify what to do with the remainder of the line.

L specifies that the remainder of the line is to be sent as a literal.

I is the same as **L**, but signals the $ IDENT card which is to be mailed back by the mail option.

B specifies that the rest of the line is a file name. That file is to be sent as binary cards.

F is the same as **B** except a form-feed is prepended to the file.

U specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.

M is followed by a user ID; after the job is sent, a message is mailed to the user via the *mail*(1) command to verify the sending of the job.

Any error encountered will cause the daemon to drop the job, wait up to 10 minutes, and start over. This means that an improperly constructed "df" file may cause the same job to be submitted every 10 minutes. *Lpd* is automatically initiated by the line printer command, *lpr*.

To restart *lpd* (in the case of hardware or software malfunction), it is necessary to first kill the old daemon (if it is still alive), and remove the lock file (if present), before initiating the new daemon. This is done automatically by **/etc/rc** when the system is brought up, in case there were any jobs left in the spooling directory when the system last went down.

FILES

The SYSTEM III commands *dpd* and *odpd*, with their associated options and files, are not implemented in Plexus Sys3, because they are specific to non-Plexus hardware.

SEE ALSO

lpr(1).

NAME

lphold, lprun - hold up print request, re-enable it

SYNOPSIS

lphold id
lprun id

DESCRIPTION

The *lphold* and *lprun* commands are for use with the **lp**(1) spooler. *Lphold* postpones the printing of the request whose request id is *id* until an *lprun*(1) command is received for that request. Conversely, *lprun* enables the printing of the request whose identification number is *id*, which has previously been put on hold by the *lphold* command. The request is placed last in the queue.

The request id is returned by the *lp* command and consists of a printer name, hyphen, and a number; for example, "printer-45".

Any user can *lphold* or *lprun* any other user's request.

FILES

/usr/spool/lp/*

NOTES

Lphold and *lprun* are Plexus commands. They are not part of standard SYSTEM III.

SEE ALSO

lp(1), lpsched(1M).

NAME

lpr – line printer spooler

SYNOPSIS

lpr [option ...] [name ...]

DESCRIPTION

Lpr causes the named files to be queued for printing on a line printer. If no names appear, the standard input is assumed; thus *lpr* may be used as a filter.

The following options may be given (each as a separate argument and in any order) before any file name arguments:

–c Makes a copy of the file to be sent before returning to the user.
–r Removes the file after sending it.
–m When printing is complete, reports that fact by *mail*(1).
–n Does not report the completion of printing by *mail*(1). This is the default option.
–b Does not print banner at beginning of output.

FILES

/etc/passwd	user's identification and accounting data.
/usr/lib/lpd	line printer daemon.
/usr/spool/lpd/*	spool area.

NOTES

If absolutely necessary, to kill a spool print job, do the following:

1. Get the process id of *lpr* using **ps**(1).
2. Kill, using **kill**(1), the **lpr** process, using the command *kill -9 <process id>*.
3. If the file **/usr/spool/lpd/lock** exists, remove it using **rm**(1).
4. List the directory **/usr/spool/lpd**. Note that it contains files whose names are prefixed with the letters 'cf' and 'df' (or 'tf'). These files come in pairs, one 'df' or 'tf' file for every 'cf' file. You can tell which are the pairs because their filenames end with the same string of digits. Look at the 'cf' files using **cat**(1). Find the one that corresponds to the job you want to kill and remove it. Find the 'df' or 'tf' file with the same last digits, and remove that one too. To help you in your search for the right files, the last digits of the relevant file names will be identical or very close to the process id of the *lpr* job.
5. To get the daemon running again, queue a short print job with a command such as **lpr /etc/group**.

SEE ALSO

lpd(1C).

BUGS

Lpr cannot print files larger than 42 full 132-column pages (about 330000 characters).

Lpr of a directory creates a directory entry in **/usr/spool/lpd**. This directory references the files in the directory specified to be printed. This reference does not increase the link count of these files; thus the deletion of the directory in **/usr/spool/lpd** deletes the files in the original directory.

NAME

lpsched, lpshut, lpmove – start/stop the LP request scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

Lpsched schedules requests taken by *lp*(1) for printing on line printers.

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp (1)* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1M)) for the new destination when moving requests.

FILES

/usr/spool/lp/*

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1).

NAME

lpstat – print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *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"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat –o
```

prints the status of all output requests.

–a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

–c[*list*] Print class names and their members. *List* is a list of class names.

–d Print the system default destination for *lp*.

–o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

–p[*list*] Print the status of printers. *List* is a list of printer names.

–r Print the status of the LP request scheduler

–s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

–t Print all status information.

–u[*list*] Print status of output requests for users. *List* is a list of login names.

–v[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

enable(1), lp(1).

NAME

ls - list contents of directories

SYNOPSIS

ls [**-logtasdrucif**] 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, but file arguments are processed before directories and their contents. There are several options:

-l List 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.

-o The same as **-l**, except that the group is not printed.

-g The same as **-l**, except that the owner is not printed.

-t Sort by time of last modification (latest first) instead of by name.

-a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.

-s Give size in 1024-byte blocks (including indirect blocks) for each entry.

-d If argument is a directory, list only its name; often used with **-l** to get the status of a directory.

-r Reverse the order of sort to get reverse alphabetic or oldest first, as appropriate.

-u Use time of last access instead of last modification for sorting (with the **-t** option) and/or printing (with the **-l** option).

-c Use time of last modification of the inode (mode, etc.) instead of last modification of the file for sorting (**-t**) and/or printing (**-l**).

-i For each file, print the i-number in the first column of the report.

-f Force each argument to be interpreted as a directory and list 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 10 characters that are interpreted as follows:

The first character is:

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 fifo (a.k.a. "named pipe") special file;
- if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three 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 three 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*(1) for the meaning of this mode. The indications of set-ID and 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	to get user IDs for **ls -l** and **ls -o**.
/etc/group	to get group IDs for **ls -l** and **ls -g**.

NOTES

Plexus provides a standalone version of *ls* in addition to the one that runs under Sys3.

SEE ALSO

chmod(1), find(1).

BUGS

The "*-g*" and "*-o*" options are incompatible.

NAME

m4 - macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

-e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.

-s Enable line sync output for the C preprocessor (#line ...)

-B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

-H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

-S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

-T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **-D** or **-U** flags:

-D*name*[=*val*]
Defines *name* to *val* or to null in *val*'s absence.

-U*name*
undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The **(** must immediately follow the name of the macro. If a defined macro name is not followed by a **(**, it is deemed to have no arguments. Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **$***n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is

like $*, but each argument is quoted (with the current quotes).

undefine removes the definition of the macro named in its argument.

defn returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef like *define*, but saves any previous definition.

popdef removes current definition of its argument(s), exposing the previous one if any.

ifdef if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX versions of *m4*.

shift returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

changequote change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., ` ').

changecom change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

divert *m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum returns the value of the current output stream.

dnl reads and discards characters up to and including the next new-line.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr returns the value of its argument decremented by 1.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to *include*, except that it says nothing if the file is inaccessible.
syscmd	executes the UNIX command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd*.
maketemp	fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup()')
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.
traceon	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.

NAME

mail, rmail - send mail to users or read mail

SYNOPSIS

mail [**-rpq**] [**-f** file]

mail persons

rmail persons

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input to determine the disposition of the message:

<new-line>	Go on to next message.
+	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
-	Go back to previous message.
s [*files*]	Save message in the named *files* (**mbox** is default).
w [*files*]	Save message, without its header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons* (yourself is default).
q	Put undeleted mail back in the *mailfile* and stop.
EOT (control-d)	Same as **q**.
x	Put all mail back in the *mailfile* unchanged and stop.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

The optional arguments alter the printing of the mail:

-r causes messages to be printed in first-in, first-out order.

-p causes all mail to be printed without prompting for disposition.

-q causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.

-f*file* causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a >. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the **dead.letter** will be saved to allow editing and resending.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will interpret that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in he is informed of the presence of mail, if any.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*	incoming mail for user *; *mailfile*
$HOME/mbox	saved mail
$MAIL	*mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), uucp(1C), write(1).

BUGS

Race conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

MAKE(1)

NAME

make - maintain, update, and regenerate groups of programs

SYNOPSIS

make [**-f** makefile] [**-p**] [**-i**] [**-k**] [**-s**] [**-r**] [**-n**] [**-b**] [**-e**] [**-m**] [**-t**] [**-q**] [**-d**] [names]

DESCRIPTION

The following is a brief description of all options and some special names:

-f *makefile* Description file name. *Makefile* is assumed to be the name of a description file. A file name of - denotes the standard input. The contents of *makefile* override the built-in rules if they are present.

-p Print out the complete set of macro definitions and target descriptions.

-i Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.

-k Abandon work on the current entry, but continue on other branches that do not depend on that entry.

-s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.

-r Do not use the built-in rules.

-n No execute mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed.

-b Compatibility mode for old makefiles.

-e Environment variables override assignments within makefiles.

-m Print a memory map showing text, data, and stack. This option is a no-operation on systems without the *getu* system call.

-t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

-d Debug mode. Print out detailed information on files and times examined.

-q Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

.DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT Same effect as the **-s** option.

.IGNORE Same effect as the **-i** option.

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **-f** option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **-**, the standard input is taken. More than one **-f** makefile argument pair may appear.

Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell

commands to be executed to update the target. The first line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the first character of the command is @. The **-n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **-i** option is present, or the entry **.IGNORE:** appears in *makefile*, or if the line specifying the command begins with <tab><hyphen>, the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old makefiles (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null) command associated with them. The previous version of *make* assumed if no command was specified explicitly that the command was null.

Interrupt and quit cause the target to be deleted unless the target depends on the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **-f**, **-p**, and **-d**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **-n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would have been executed. This is because the **-n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form *string1* = *string2* are macro definitions. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of

substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out of date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc -c -O $*.c
```

or:

```
.c.o:
        cc -c -O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

The only peculiarity in this output is the **(null)** string which *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see *sccsfile*(5)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix it is incompatible with *make*'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e. **.c:**) is the definition of how to build *x* from *x*.**c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite.

The default list is:

.SUFFIXES: .o .c .y .l .s

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1) respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parenthesis, it is assumed to be an archive library, the string within parenthesis referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form **.*XX*.a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up to date
.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $?  @echo lib is now up to date
```

```
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out of date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile
s.[Mm]akefile

SEE ALSO

sh(1), mk(8).
Make-A Program for Maintaining Computer Programs by S. I. Feldman.
An Augmented Version of Make by E. G. Bradford.

DIAGNOSTICS

The message "$! nulled, predecessor circle" means the makefile has a circular dependency chain.

BUGS

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*. The syntax **(lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c~)** doesn't work.

NAME

man - print entries in this manual

SYNOPSIS

man [options] [section] titles

DESCRIPTION

Man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are:

-t	Typeset the entry in the default format (8.5''×11'').
-s	Typeset the entry in the small format (6''×9'').
-T4014	Display the typeset output on a Tektronix 4014 terminal using *tc*(1).
-Ttek	Same as **-T4014**.
-Tst	Print the typeset output on the MHCC STARE facility (see *gcat*(1C)).
-Tvp	Print the typeset output on a Versatec printer using *vpr*(1); this option is not available at all UNIX sites.
-T*term*	Format the entry using *nroff*(1) and print it on the standard output (usually, the terminal); *term* is the terminal type (see *term*(7) and the explanation below); for a list of recognized values of *term*, type **help term2**. The default value of *term* is **450**.
-w	Print on the standard output only the *path names* of the entries, relative to **/usr/man**, or to the current directory for **-d** option.
-d	Search the current directory rather than **/usr/man**; requires the full file name (e.g., **cu.1c**, rather than just **cu**).
-12	Indicates that the manual entry is to be produced in 12-pitch. May be used when **$TERM** (see below) is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)
-c	Causes *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of **300**, **300s**, **450**, **37**, **4000A**, **382**, **4014**, **tek**, **1620**, and **X**.
-y	Causes *man* to use the non-compacted version of the macros.

The above *options* are mutually exclusive, except that the **-s** option may be used in conjunction with the first four **-T** options above. Any other *options* are passed to *troff*(1), *nroff*(1), or the *man*(7) macro package.

When using *nroff*(1), *man* examines the environment variable **$TERM** (see *environ*(7)) and attempts to select options to *nroff*(1), as well as filters, that adapt the output to the terminal being used. The **-T***term* option overrides the value of **$TERM**; in particular, one should use **-Tlp** when sending the output of *man* to a line printer.

Section may be changed before each *title*.

As an example:

```
man man
```

would reproduce on the terminal this entry, as well as any other entries named *man* that may exist in other sections of the manual, e.g., *man*(7).

If the first line of the input for an entry consists solely of the string:

```
'\" x
```

where *x* is any combination of the three characters **c**, **e**, and **t**, and where there is exactly one blank between the double quote (**"**) and *x*, then *man* will preprocess its input through the appropriate combination of *cw*(1), *eqn*(1) or *neqn*(1), and *tbl*(1), respectively.

FILES

/usr/man/man[1-8]/*
/usr/man/local/man[1-8]/*

SEE ALSO

cw(1), eqn(1), gcat(1C), tbl(1), tc(1), troff(1), environ(7), man(7), term(7).

BUGS

All entries are supposed to be reproducible either on a typesetter or on a terminal. However, on a terminal some information is necessarily lost.

NAME

mesg - permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user 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(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

MKDIR(1)

NAME

mkdir - make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777. Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

rm(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs special blocks[:inodes] [gap blocks]
/etc/mkfs special proto [gap blocks]

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. If the second argument is given as a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of blocks divided by 4.

If the second argument is a file name that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the created file system. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **-bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or - to specify set-user-id mode or not. The third is **g** or - for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions (see *chmod*(1)).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a path name whence the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *blocks* can be specified.

NOTES

Plexus provides a standalone version of *mkfs* in addition to the one that runs under Sys3.

Be sure you have done a *mknod*(1) for the special device *before* you run *mkfs*.

Because *fsck*(1M) cannot check very large file systems, it is recommended that file system sizes on Z8000 systems not exceed 68K blocks. On MC68000 systems, they should not exceed 140K blocks.

SEE ALSO

fsck(1M), dir(5), fs(5).

BUGS

If a prototype is used, it is not possible to initialize a file larger than 64K bytes, nor is there a way to specify links.

Since lines beginning with a colon (:) are treated as comments, there is no way to specify a file name whose first character is a colon.

NAME

mknod - build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor
/etc/mknod name **p**

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry. In the first case, the second 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. They have to be dug out of the system source file **conf.c**.

Mknod can also be used to create fifo's (a.k.a named pipes) (second case in *SYNOPSIS* above).

SEE ALSO

mknod(2).

NAME

mkstr - create an error message file by massaging C source

SYNOPSIS

/usr/plx/mkstr [-] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can reduce the size of programs with large numbers of error diagnostics. It can also reduce system overhead in running the program, since the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical usage of *mkstr* would be

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx.*

To process the error messages in the source to the message file *mkstr* keys on the string 'error(" ' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, and the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file. The pointer can then be used to retrieve the message, i.e.:

```
char    efilname[] = "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

The optional - causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

NOTES

This program is based on a similar one from the University of California at Berkeley.

SEE ALSO

lseek(2), xstr(1)

BUGS

All the arguments except the name of the file to be processed are unnecessary.

NAME

mm - print out documents formatted with the MM macros

SYNOPSIS

mm [options] [files]

DESCRIPTION

Mm can be used to type out documents using *nroff*(1) and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn*(1) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff*(1) and MM are generated, depending on the options selected.

Options for *mm* are given below. Any other arguments or flags (e.g., **-rC3**) are passed to *nroff*(1) or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

-T*term* Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **$TERM** from the environment (see *profile*(5) and *environ*(7)) as the value of *term*, if **$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.

-12 Indicates that the document is to be produced in 12-pitch. May be used when **$TERM** is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)

-c Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000A**, **382**, **4014**, **tek**, **1620**, and **X**.

-e Causes *mm* to invoke *neqn*(1).

-t Causes *mm* to invoke *tbl*(1).

-E Invokes the **-e** option of *nroff*(1).

-y Causes *mm* to use the non-compacted version of the macros (see *mm*(7)).

As an example (assuming that the shell variable **$TERM** is set in the environment to **450**), the two command lines below are equivalent:

```
mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3
```

Mm reads the standard input when **-** is specified instead of any file names. (Mentioning other files together with **-** leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm -
```

HINTS

1. *Mm* invokes *nroff*(1) with the **-h** flag. With this flag, *nroff*(1) assumes that the terminal has tabs set every 8 character positions.
2. Use the **-o***list* option of *nroff*(1) to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **-e**, **-t**, and **-** options, *together* with the **-o***list* option of *nroff*(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the **-s** option of *nroff*(1) (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The **-s** option of *nroff*(1) does not work with the **-c** option of *mm*, or if *mm* automatically invokes *col*(1) (see **-c** option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you'll get (often subtle) garbage; however, if you are redirecting output into a file, use the **-T37** option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

col(1), env(1), eqn(1), greek(1), mmt(1), nroff(1), tbl(1), profile(5), mm(7), term(7).
MM-Memorandum Macros by D. W. Smith and J. R. Mashey.
Typing Documents with MM by D. W. Smith and E. M. Piskorik.

DIAGNOSTICS

"mm: no input file" if none of the arguments is a readable file and *mm* is not used as a filter.

NAME

mmchek - check usage of mm macros and eqn delimiters

SYNOPSIS

mmchek [files]

DESCRIPTION

Mmchek is a program for checking the contents of the named *files* for errors in the use of Memorandum Macros (see *mm*(1)) and some *eqn*(1) constructions. Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

SEE ALSO

eqn(1), mm(1), mmt(1).
MM-Memorandum Macros by D. W. Smith and J. R. Mashey.

DIAGNOSTICS

Unreadable files cause the message "Cannot open *file-name*". The remaining output of the program is diagnostic of the source file.

BUGS

This is an experimental version of *mmchek*. *Mmchek* may be fully supported in the future.

NAME

mmt, mvt - typeset documents, view graphs, and slides

SYNOPSIS

mmt [options] [files]

mvt [options] [files]

DESCRIPTION

These two commands are very similar to *mm*(1), except that they both typeset their input via *troff*(1), as opposed to formatting it via *nroff*(1); *mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by *tbl*(1) and/or *eqn*(1). The proper pipelines and the required arguments and flags for *troff*(1) and for the macro packages are generated, depending on the options selected.

Options are given below. Any other arguments or flags (e.g., **-rC3**) are passed to *troff*(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

-e Causes these commands to invoke *eqn*(1).

-t Causes these commands to invoke *tbl*(1).

-Tst Directs the output to the MH STARE facility.

-Tvp Directs the output to a Versatec printer via the *vpr*(1) spooler; this option is not available at all UNIX sites.

-T4014 Directs the output to a Tektronix 4014 terminal via the *tc*(1) filter.

-Ttek Same as **-T4014**.

-a Invokes the **-a** option of *troff*(1).

-y Causes *mmt* to use the non-compacted version of the macros (see *mm*(7)). No effect for *mvt*.

These commands read the standard input when - is specified instead of any file names.

Mvt is just a link to *mmt*.

HINT

Use the **-o***list* option of *troff*(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the **-e**, **-t**, and - options, *together* with the **-o***list* option of *troff*(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

SEE ALSO

env(1), eqn(1), mm(1), tbl(1), tc(1), troff(1), profile(5), environ(7), mm(7), mv(7).
MM-Memorandum Macros by D. W. Smith and J. R. Mashey.
Typing Documents with MM by D. W. Smith and E. M. Piskorik.
A Macro Package for View Graphs and Slides by T. A. Dolotta and D. W. Smith (in preparation).

DIAGNOSTICS

"m[mv]t: no input file" if none of the arguments is a readable file and the command is not used as a filter.

NAME

more, page - file perusal filter for crt viewing

SYNOPSIS

/usr/plx/more [**-d**] [**-f**] [**-l**] [*-n*] [*+linenumber*] [*+/pattern*] [name ...]

page [**-d**] [**-f**] [**-l**] [*-n*] [*+linenumber*] [*+/pattern*] [name ...]

DESCRIPTION

More is a filter that allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilites are enumerated later.

The command line options are:

-n An integer that is the size (in lines) of the window that *more* will use instead of the default.

-d *More* will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

-f This 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 which would ordinarily occupy screen postions, but which 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 Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

+linenumber
Start up at *linenumber.*

+/pattern
Start up two lines before the line containing the regular expression *pattern.*

If the program is invoked as *page,* then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and *k* - 1 rather than *k* - 2 lines are printed in each screenful, where *k* is the number of lines the terminal can display.

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.

If *more* is reading from a file, rather than a pipe, then 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 typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i <space>
display *i* more lines, (or another screenful if no argument is given)

^D display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d same as ^D (control-D)

*i*z same as typing a space except that *i*, if present, becomes the new window size.

*i*s skip *i* lines and print a screenful of lines

*i*f skip *i* screenfuls and print a screenful of lines

q or Q Exit from *more.*

= Display the current line number.

v Start up the editor *vi* at the current line.

h Help command; give a description of all the *more* commands.

i/expr search 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 screenful 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 search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go 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
invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *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 is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q
exit from *more* (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). *More* will stop sending output, and will display the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *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
```

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

script(1)

NAME

mount, umount - mount and dismount file system

SYNOPSIS

/etc/mount [special directory [**-r**]]

/etc/umount special

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The *directory* must exist already; it becomes the name of the root of the newly mounted file system.

These commands maintain a table of mounted devices. If invoked with no arguments, *mount* prints the table.

The optional last argument indicates that the file is to be mounted read-only. Physically write-protected and magnetic tape file systems must be mounted in this way or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

FILES

/etc/mnttab mount table

SEE ALSO

mount(2), mnttab(5).

DIAGNOSTICS

Mount issues a warning if the file system to be mounted is currently mounted under another name, or if the file system is mounted by another name than its default name assigned via the **/etc/labelit** command.

Umount complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory.

BUGS

Some degree of validation is done on the file system, however it is generally unwise to mount garbage file systems.

MVDIR(1M)

NAME

mvdir - move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

Mvdir renames directories within a file system. *Dirname* must be a directory; *name* must not exist. Neither name may be a sub-set of the other (**/x/y** cannot be moved to **/x/y/z**, nor vice versa).

Only super-user can use *mvdir*.

SEE ALSO

mkdir(1).

NAME

ncheck - generate names from i-numbers

SYNOPSIS

ncheck [**-i** numbers] [**-a**] [**-s**] [file-system]

DESCRIPTION

Ncheck with no argument generates a path name vs. i-number list of all files on a set of default file systems. Names of directory files are followed by /.. The **-i** option reduces the report to only those files whose i-numbers follow. The **-a** option allows printing of the names . and .., which are ordinarily suppressed. The **-s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a path name beginning with **...** denotes a loop.

NAME

newgrp - log in to a new group

SYNOPSIS

newgrp [group]

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to *login*(1). 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; in effect it changes the group identification back to the caller's original group.

A password is demanded if the group has a password and the user himself does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

When most users log in, they are members of the group named **other**.

FILES

/etc/group
/etc/passwd

SEE ALSO

login(1), group(5).

BUGS

There is no convenient way to enter a password into **/etc/group**.
Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

news - print news items

SYNOPSIS

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. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(5), environ(7).

NAME

nice - run a command at low priority

SYNOPSIS

nice [-increment] command [arguments]

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **--10**.

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl - line numbering filter

SYNOPSIS

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 are signaled by input lines containing nothing but the following character(s):

Line contents	*Start of*
\:\:\:	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 file name. 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 Do 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**. *Width* can't be greater than 99.

-n*format* *Format* is the line numbering format. Recognized values are: **l**, left justified, leading zeroes supressed; **rn**, right justified, leading zeroes supressed; **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(1).

NAME

nm - print name list

SYNOPSIS

nm [**-gnoprsu**] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in **a.out** are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

-g Print only global (external) symbols.

-n Sort numerically rather than alphabetically.

-o Prepend file or archive element name to each output line rather than only once. This option can be used to make piping to *grep*(1) more meaningful.

-p Don't sort; print in symbol-table order.

-r Sort in reverse order.

-s Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next highest value). This difference is the value printed. This flag turns on **-g** and **-n** and turns off **-u** and **-p**.

-u Print only undefined symbols.

SEE ALSO

ar(1), a.out(5), ar(5).

NAME

node - enable or disable foreign hosts

SYNOPSIS

node [**-d**] [nodename ...] [**-n** namelist]

node [**-e**] [nodename ...] [**-n** namelist]

DESCRIPTION

This command allows a node to enable or disable *rmount*s to or from other nodes. If another node is enabled, *rmount*s are permitted to and from that node; if another node is disabled, any current connections are terminated gracefully and further *rmount*s are not permitted.

The *node -d* command should be used when *rumount* is not possible, e.g., when node1 is *rmount*ed on node2 and needs to disconnect, but there is no one at node2 to perform the *rumount*. It also prevents multiple *rumount*s.

If *rmount*s are usually desired, *node -e* commands should be placed in **/etc/rc**.

When *node* is invoked without an argument, it lists the nodenames in the configuration file (**/usr/lib/nos/D-hosts**) and indicates if the node is enabled or disabled.

The following options are permitted:

-d With *nodename*(s), disable the specified node(s). No new connections to these nodes are permitted; all existing connections are broken. With no argument, disable all nodes.

-e With *nodename*(s), enable specified node(s), and allow new connections to the designated node(s). With no arguments, enables all nodes.

-n Analogous to the "-n" option of *ps*, this allows querying a kernel (named by *namelist*) whose name is different from the default (**/sys3**).

FILES

/sys3
/dev/kmem

SEE ALSO

rmount(1M), D-hosts(5).

NOTES

This command is available on the Plexus Network Operating System (NOS) only.

NAME

nohup - run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, it will be sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

NOTES

Csh has a *nohup* command that is incompatible with this one. *Csh* users must specify the whole pathname of this *nohup* command (**/usr/bin/nohup**) in order to access it.

SEE ALSO

nice(1), signal(2).

NAME

od - octal dump

SYNOPSIS

od [-bcdox] [file] [[+]offset[**.][b]]**

DESCRIPTION

Od dumps *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 Interpret bytes in octal.

-c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**\0**, backspace=**\b**, form-feed=**\f**, new-line=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.

-d Interpret words in decimal.

-o Interpret words in octal.

-x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. 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 of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

NOTES

Plexus provides a standalone version of *od* in addition to the one that runs under Sys3.

SEE ALSO

adb(1).

NAME

openup - keep open key directories and files

SYNOPSIS

/etc/openup files [-r files] [-w files] [-rw files]

DESCRIPTION

Openup is a daemon that opens and keeps open key directories and files. It is normally invoked by **/etc/rc**. Key directories and files should be kept open because (1) inodes are thereby kept in memory, resulting in more efficient access; and (2) the *stty* modes of serial port lines can be set and maintained across several opens.

Openup options include:

-r Subsequent files are opened read-only.

-w Subsequent files are opened write-only.

-rw Subsequent files are opened read/write.

The default mode is to open a file read-only.

Some of the directories and files you may want to *openup* are:

/
/bin
/lib
/etc
/tmp
/usr/bin
/usr/lib
/usr/tmp
/dev/lp

NOTES

This is a Plexus command. It is not part of standard SYSTEM III.

NAME

pack, pcat, unpack - compress and expand files

SYNOPSIS

pack [-] name ...

pcat name ...

unpack name ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.**z** with the same access modes, access and modified dates, and owner as those 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 the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. 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 skewed, 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 file name 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 file name 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*(1) 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**) use the command:

```
pcat name >nnn
```

Pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (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 for the following:

a file with the "unpacked" name already exists;
if the unpacked file cannot be created.

NAME

passwd - change login password

SYNOPSIS

passwd name

DESCRIPTION

This command changes (or installs) a password associated with the login *name*.

The program prompts for the old password (if any) and then for the new one (twice). The caller must supply these. New passwords should be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. Only the first eight characters of the password are significant.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password. Only the super-user can create a null password.

The password file is not changed if the new password is the same as the old password, or if the password has not "aged" sufficiently; see *passwd*(5)).

FILES

/etc/passwd

SEE ALSO

login(1), crypt(3C), passwd(5).

NAME

paste - merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste -d list file1 file2 ...
paste -s [-d list**]** 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). If you will, it is the counterpart of *cat*(1) 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 file name.

The meanings of the options are:

-d Without this option, the new-line 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 new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\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 Merge 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 new-line.

- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls | paste -d" " -	list directory in one column
ls | paste - - - -	list directory in four columns
paste -s -d"\t\n" file	combine pairs of lines into lines

SEE ALSO

grep(1), cut(1),
pr(1): **pr -t -m**... works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

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.

NAME

pr - print files

SYNOPSIS

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, a 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. In addition, messages to the terminal, e.g., from *write* or *wall*, are disabled while printing to the terminal is in progress.

Options may appear singly or be combined in any order. Their meanings are:

+*k* Begin printing with page *k* (default is 1).

-*k* Produce *k*-column output (default is 1). The options **-e** and **-i** are assumed for multi-column output.

-a Print multi-column output across the page.

-m Merge and print all files simultaneously, one per column (overrides the **-***k*, and **-a** options).

-d Double-space the output.

-e*ck* Expand *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 eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

-i*ck* In *output*, replace white space 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 eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

-n*ck* Provide *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 non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

-w*k* Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).

-o*k* Offset 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* Set the length of a page to *k* lines (default is 66).

-h Use the next argument as the header to be printed instead of the file name.

-p Pause 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 Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a

terminal.

-r Print no diagnostic reports on failure to open files.

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

-sc Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr -e9 -t <file1 >file2
```

FILES

/dev/tty* used to suspend messages while printing is in progress

SEE ALSO

cat(1).

NAME

printenv – print out the environment

SYNOPSIS

/usr/plx/printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

If a *name* is specified and it is not defined in the environment, *printenv* returns exit status 1, else it returns status 0.

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

sh(1), environ(5), csh(1).

NAME

prof - display profile data

SYNOPSIS

prof [**-v**] [**-a**] [**-l**] [-low [-high]] [file]

DESCRIPTION

Prof interprets the file **mon.out** produced by the *monitor*(3C) subroutine. Under default modes, the symbol table in the named object file (**a.out** default) is read and correlated with the **mon.out** profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the **-a** option is used, all symbols are reported rather than just external symbols. If the **-l** option is used, the output is listed by symbol value rather than decreasing percentage.

If the **-v** option is used, all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the *tplot*(1G) filters. The optional arguments *low* and *high*, by default 0 and 100, cause a selected percentage of the profile to be plotted with accordingly higher resolution.

In order for the number of calls to a routine to be tallied, the **-p** option of *cc* must have been given when the file containing the routine was compiled. This option also arranges for the **mon.out** file to be produced automatically.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

cc(1), tplot(1G), profil(2), monitor(3C).

BUGS

Beware of quantization errors.

NAME

prfld, prfstat, prfdc, prfsnap, prfpr - operating system profiler

SYNOPSIS

/etc/prfld [namelist]
/etc/prfstat [**on** | **off**]
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [namelist]]

DESCRIPTION

Prfld, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* form a system of programs to facilitate an activity study of the UNIX operating system.

Prfld is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *namelist*.

Prfstat is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. *Prfstat* will also reveal the number of text addresses being measured.

Prfdc and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *Prfdc* will store the counters into *file* every *period* minutes and will turn off at *off_hour*. *Prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

Prfpr formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

FILES

/dev/prf interface to profile data and text addresses
/sys3 default for namelist file

SEE ALSO

prf(4).

NAME

prs - print an SCCS file

SYNOPSIS

prs [**-d**[dataspec]] [**-r**[SID]] [**-e**] [**-l**] [**-a**] files

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

-d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

-r[*SID*] Used to specify the *S*CCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

-e Requests information for all deltas created *earlier* than and including the delta designated via the **-r** keyletter.

-l Requests information for all deltas created *later* than and including the delta designated via the **-r** keyletter.

-a Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, i.e., delta type = *D*, deltas. If the **-a** keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by **\t** and carriage return/new-line is specified by **\n**.

PRS(1)

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:.:Tm:.:Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:
xyz
131
abc

prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case:*

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the **-a** keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

PS(1)

NAME

ps - report process status

SYNOPSIS

ps [options]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. Otherwise, the information that is displayed is controlled by the following *options*:

-e Print information about all processes.

-d Print information about all processes, except process group leaders.

-a Print information about all processes, except process group leaders and processes not associated with a terminal.

-f Generate a *full* listing. (Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.

-l Generate a *long* listing. See below.

-c *corefile* Use the file *corefile* in place of **/dev/kmem**.

-s *swapdev* Use the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*; a *swapdev* of **/dev/null** will cause the user block to be zeroed out.

-n *namelist* The name of the operating system being executed. (**/sys3** is the default).

-t *tlist* Restrict 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* Restrict listing to data about processes whose process ID numbers are given in *plist*, where *plist* is in the same format as *tlist*.

-u *ulist* Restrict 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 will be printed unless the **-f** option is used, in which case the login name will be printed.

-g *glist* Restrict 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*.

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) Flags (octal and additive) associated with the process:

- 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;
- W waiting;
- R running;
- I intermediate;
- Z terminated;
- T stopped.

UID	(f,l)	The user ID number of the process owner; the login name is printed under the **-f** option.
PID	(all)	The process ID of the process; it is possible to kill a process if you know this datum.
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.
ADDR	(l)	The memory address of the process, if resident; otherwise, the disk address.
SZ	(l)	The size in 512-byte increments of the unshared portion of the core image of the process. Due to memory management hardware, this is always a multiple of 4 on the Z8000 and 8 on the MC68000.
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.
COMD	(l)	The command name; the full command name and its arguments are printed under the **-f** option. This heading is 'COMMAND' in non-*long* listings.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **-f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **-f** option, is printed in square brackets.

FILES

/sys3	system namelist
/dev/kmem	kernel memory
/dev	searched to find swap device and terminal ("tty") names.

NOTES

Ps reports PID 0 as COMD or COMMAND 'swapper'. This is the system scheduler and idle loop. When the system has no work to do, TIME is charged to this PID.

SEE ALSO

kill(1), nice(1).

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

NAME

ptx - permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where **.xx** is assumed to be an *nroff* or *troff*(1) macro provided by the user. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following options can be applied:

-f Fold upper and lower case letters for sorting.

-t Prepare the output for the phototypesetter.

-w *n* Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.

-g *n* Use the next argument, *n*, as the number of characters that *ptx* will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

-o *only* Use as keywords only the words given in the *only* file.

-i *ignore* Do not use as keywords any words given in the *ignore* file. If the **-i** and **-o** options are missing, use **/usr/lib/eign** as the *ignore* file.

-b *break* Use the characters in the *break* file to separate words. Tab, new-line, and space characters are *always* used as break characters.

-r Take any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

FILES

/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.
Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

NAME

pwck, grpck - password/group file checkers

SYNOPSIS

pwck [file]
grpck [file]

DESCRIPTION

Pwck scans the password file and notes 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 criteria for determining a valid login name are taken from *Setting Up UNIX*. The default password file is **/etc/passwd**.

Grpck 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

group(5), passwd(5).
Setting Up UNIX.

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

PWD(1)

NAME

pwd - working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX programming counselor.

NAME

ratfor - rational Fortran dialect

SYNOPSIS

ratfor [options] [files]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:
{ statement; statement; statement }

decision-making:
if (condition) statement [**else** statement]
switch (integer value) {
case integer: statement
...
[**default**:] statement
}

loops:
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [**until** (condition)]
break
next

and some syntactic sugar to make programs easier to read and write:

free form input:
multiple statements/line; automatic continuation

comments:
this is a comment.

translation of relationals:
>, >=, etc., become **.GT.**, **.GE.**, etc.

return expression to caller from function:
return (expression)

define:
define *name replacement*

include:
include *file*

The option **-h** causes quoted strings to be turned into **27H** constructs. The **-C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **-6x** makes the continuation character **x** and places it in column 6.

Ratfor is best used with *f77*(1), Plexus product number 4214A (for P/35 and P/60) and 4108A (for P/25 and P/40).

SEE ALSO

efl(1).
B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

reform - reformat text file

SYNOPSIS

reform [tabspec1 [tabspec2]] [+**b**n] [+**e**n] [+**f**] [+**i**n] [+**m**n] [+**p**n] [+**s**] [+**t**n]

DESCRIPTION

Reform reads each line of the standard input file, reformats it, and then writes it to the standard output. Various combinations of reformatting operations can be selected, of which the most common involve rearrangement of tab characters. It is often used to trim trailing blanks, truncate lines to a specified length, or prepend blanks to lines.

Reform first scans its arguments, which may be given in any order. It then processes its input file, performing the following actions upon each line, in the order given:

- A line is read from the standard input.
- If +**s** is given, all characters up to the first tab are stripped off and saved for later addition to the end of the line. Presumably, these characters comprise an "SCCS SID" produced by *get*(1).
- The line is expanded into a tabless form, by replacing tabs with blanks according to the *input* tab specification *tabspec1*.
- If +**p***n* is given, *n* blanks are prepended to the line.
- If +**t***n* is given, the line is truncated to a length of *n* characters.
- All trailing blanks are now removed.
- If +**e***n* is included, the line is extended out with blanks to the length of *n* characters.
- If +**s** is given, the previously-saved "SCCS SID" is added to the end of the line.
- If +**b***n* is given, the *n* characters at the beginning of the line are converted to blanks, if and only if all of them are either digits or blanks.
- If +**m***n* is included, the line is moved left, i.e., *n* characters are removed from the beginning of the line.
- The line is now contracted by replacing some blanks with tab characters according to the list of tabs indicated by the *output* tab specification *tabspec2*, and is written to the standard output file. Option +**i** controls the method of contraction (see below).

The various arguments accepted by *reform* are as follows:

tabspec1

describes the tab stops assumed for the input file. This tab specification may take on any of the forms described in *tabs*(1). In addition, the operand **-** indicates that the tab specification is to be found in the first line read from the standard input. If no legal tab specification is found there, **-8** is assumed. If *tabspec1* is omitted entirely, **-** is assumed.

tabspec2

describes the tabs assumed for the output file. It is interpreted in the same way as *tabspec1*, except that omission of *tabspec2* causes the value of *tabspec1* to be used for *tabspec2*.

The remaining arguments are all optional and may be used in any combination, although only a few combinations make much sense. Specifying an argument causes an action to be performed, as opposed to the usual default of not performing the action. Some options include numeric values, which also have default values. Option actions are applied to each line in the order described above. Any line length mentioned applies to the length of a line just before the

execution of the option described, and the terminating new-line is never counted in the line length.

+b*n* causes the first *n* characters of a line to be converted to blanks, if and only if those characters include only blanks and digits. If *n* is omitted, the default value is 6, which is useful in deleting sequence numbers from COBOL programs.

+e*n* causes each line shorter than *n* characters to be extended out with blanks to that length. Omitting *n* implies a default value of 72. This option is useful for those rare cases in which sequence numbers need to be added to an existing unnumbered file. The use of **$** in editor regular expressions is more convenient if all lines have equal length, so that the user can issue editor commands such as:
s/$00001000/

+f causes a format line to be written to the standard output, preceding any other lines written. See *fspec(5)* for details regarding format specifications. The format line is taken from *tabspec2*, i.e., the line normally appears as follows:
<:t-*tabspec2* **d:>**

If *tabspec2* is of the form *--file-name* (i.e., an indirect reference to a tab specification in the first line of the named file), then that tab specification line is written to the standard output.

+i*n* controls the technique used to compress interior blanks into tabs. Unless this option is specified, any sequence of 1 or more blanks may be converted to a single tab character if that sequence occurs just before a tab stop. This causes no problems for blanks that occur before the first nonblank character in a line, and it is always possible to convert the result back to an equivalent tabless form. However, occasionally an interior blank (one occurring after the first nonblank) is converted to a tab when this is not intended. For instance, this might occur in any program written in a language utilizing blanks as delimiters. Any single blank might be converted to a tab if it occurred just before a tab stop. Insertion or deletion of characters preceding such a tab may cause it to be interpreted in an unexpected way at a later time. If the **+i** option is used, no string of blanks may be converted to a tab unless there are *n* or more contiguous blanks. The default value is 2. Note that leading blanks are always converted to tabs when possible. It is recommended that conversion of programs from non -UNIX to UNIX systems use this option.

+m*n* causes each line to be moved lift *n* characters, with a default value of 6. This can be useful for crunching COBOL programs.

+p*n* causes *n* blanks to be prepended (default of 6 if *n* is omitted). This option is effectively the inverse of **+m***n*, and is often useful for adjusting the position of *nroff*(1) output for terminals lacking both forms tractor positioning and a settable left margin.

+s is used with the **-m** option of *get*(1). The **-m** option causes *get* to prepend to each generated line the appropriate SCCS SID, followed by a tab. The **+s** option causes *reform* to remove the SID from the front of the line, save it, then add it later to the end of the line. Because **+e72** is implied by this option, the effect is to produce 80-character card images with SCCS SID in columns 73-80. Up to 8 characters of the SID are shown; if it is longer, the eighth character is replaced by * and any characters to the right of it are discarded.

+t*n* causes any line longer than *n* characters to be truncated to that length. If *n* is omitted, the length defaults to 72. Sequence numbers can thus be removed and any blanks immediately preceding them deleted.

The following illustrate typical uses of *reform*. The terms **PWB** and **OBJECT** below refer to UNIX and non- UNIX computer systems, respectively. Each arrow indicates the direction of

conversion. The character **?** indicates an arbitrary tab specification; see *tabs*(1) for descriptions of legal specifications.

OBJECT ---> PWB (i.e., manipulation of RJE output):

Note that files transferred by RJE from OBJECT to PWB materialize with format **-8**.

reform -8 -c +t +b +i <oldfile >newfile (into COBOL)
reform -8 -c3 +t +m +i <oldfile >newfile (into COBOL, crunched)

NOTE: -c3 is the preferred format COBOL; it uses the least disk space of the COBOL formats.

PWB ---> OBJECT (i.e., preparation of files for RJE submission):

reform ? -8 <oldfile >newfile (from arbitrary format into **-8**)
get -p -m sccsfile | reform +s | send ...

PWB ONLY (i.e., no involvement with other systems):

pr file | reform ? -0 <oldfile (print on terminal without hardware tabs)
reform ? -0 <oldfile >newfile (convert file to tabless format)

DIAGNOSTICS

All diagnostics are fatal, and the offending line is displayed following the message.
"line too long" a line exceeds 512 characters (in tabless form).
"not SCCS -m" a line does not have at least one tab when **+s** flag is used.
Any of the diagnostics of *tabs*(1) can also appear.

EXIT CODES

0 - normal
1 - any error

SEE ALSO

get(1), nroff(1), send(1C), tabs(1), fspec(5).

BUGS

Reform is aware of the meanings of backspaces and escape sequences, so that it can be used as a postprocessor for *nroff*. However, be warned that the **+e**, **+m**, and **+t** options only count characters, not positions. Anyone using these options on output containing backspaces or half-line motions will probably obtain unexpected results.

NAME

regcmp - regular expression compile

SYNOPSIS

regcmp [-] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp* (see *regex*(3X)) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.**i**. If the - option is used, the output will be placed in *file*.**c**. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File*.**i** files may thus be *included* into C programs, or *file*.**c** files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name    "([A-Za-z][A-Za-z0-9_]*)$0"

telno   "\({0,1}([2-9][01][1-9])$0\){0,1} *"
        "([2-9][0-9]{2})$1[ -]{0,1}"
        "([0-9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regex(3X).

NAME

restor - incremental file system restore

SYNOPSIS

restor key [arguments]

DESCRIPTION

Restor is used to read magnetic tapes dumped with the *dump* command. The *key* specifies what is to be done. *Key* is one of the characters **rRxt**, optionally combined with **f**.

f Use the first *argument* as the name of the tape instead of the default.

r or **R** The tape is read and loaded into the file system specified in *argument*. This should not be done lightly (see below). If the key is **R**, *restor* asks which tape of a multi-volume set to start on. This allows *restor* to be interrupted and then restarted (an *fsck* must be done before the restart).

x Each file on the tape named by an *argument* is extracted. The file name has all "mount" prefixes removed; for example, if **/usr** is a mounted file system, **/usr/bin/lpr** is named **/bin/lpr** on the tape. The extracted file is placed in a file with a numeric name supplied by *restor* (actually the inode number). In order to keep the amount of tape read to a minimum, the following procedure is recommended:

1. Mount volume 1 of the set of dump tapes.
2. Type the *restor* command.
3. *Restor* will announce whether or not it found the files, give the numeric name that it will assign to the file, and rewind the tape.
4. It then asks you to "mount the desired tape volume". Type the number of the volume you choose. On a multi-volume dump the recommended procedure is to mount the last through the first volumes, in that order. *Restor* checks to see if any of the requested files are on the mounted tape (or a later tape-thus the reverse order) and doesn't read through the tape if no files are. If you are working with a single-volume dump or if the number of files being restored is large, respond to the query with **1** and *restor* will read the tapes in sequential order.

t Print the date the tape was written and the date the file system was dumped from.

The **r** option should only be used to restore a complete dump tape onto a clear file system, or to restore an incremental dump tape onto a file system so created. Thus:

```
/etc/mkfs /dev/dk1 18000
restor r /dev/dk1
```

is a typical sequence to restore a complete dump. Another *restor* can be done to get an incremental dump in on top of this.

A *dump* followed by a *mkfs* and a *restor* is used to change the size of a file system.

In the standalone version of this program, a final +*n* argument advances the tape *n* files before executing the *restor*. To space forward *n* files in the online version type

```
/usr/plx/tape srcheof n
```

before typing the *restor* command.

FILES

/dev/rmt0
rst*

NOTES

This command has a standalone version. The **x** option of the standalone version does not work.

SEE ALSO

dump(1M), dumpdir(1M), fsck(1M), mkfs(1M).

DIAGNOSTICS

There are various diagnostics involved with reading the tape and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

BUGS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, *restor* doesn't use it.

NAME

rjestat - RJE status report and interactive status console

SYNOPSIS

rjestat [*host*]... [**-s***host*] [**-c***host cmd*]...

DESCRIPTION

Rjestat provides a method of determining the status of an RJE link and of simulating an IBM remote console (with UNIX features added). When invoked with no arguments, *rjestat* reports the current status of all the RJE links connected to to the UNIX system. The options are:

host Print the status of the line to *host*. *Host* is the pseudonym for a particular IBM system. It can be any name that corresponds to one in the first column of the RJE configuration file.

-s*host* After all the arguments have been processed, start an interactive status console to *host*.

-c*host cmd* Interpret *cmd* as if it were entered in status console mode to *host*. See below for the proper format of *cmd*.

In status console mode, *rjestat* prompts with the host pseudonym followed by : whenever it is ready to accept a command. Commands are terminated with a new-line. A line that begins with **!** is sent to the UNIX shell for execution. A line that begins with the letter **q** terminates *rjestat*. All other input lines are assumed to have the form:

ibmcmd [*redirect*]

Ibmcmd is any IBM JES or HASP command. Only the super-user or **rje** login can send commands other than display or inquiry commands. *Redirect* is a pipeline or a redirection to a file (e.g., "> file" or " | grep ..."). The IBM response is written to the pipeline or file. If *redirect* is not present, the response is written to the standard output of *rjestat*.

An interrupt signal (DEL or BREAK) will cancel the command in progress and cause *rjestat* to return to the command input mode.

EXAMPLE

The following command reports the status of all the card readers attached to host A, remote 5. JES2 is assumed.

```
rjestat -cA '$du,rmt5 | grep RD'
```

DIAGNOSTICS

The message "RJE error: ..." indicates that *rjestat* found an inconsistency in the RJE system. This may be transient but should be reported to the site administrator.

FILES

/usr/rje/lines RJE configuration file

resp host response file that exists in the RJE subsystem directory (e.g., **/usr/rje1**).

SEE ALSO

send(1C), rje(8).
OS/VS2 HASP II Version 4 Operator's Guide, IBM SRL #GC27-6993.
Operator's Library: OS/VS2 Reference (JES2), IBM SRL #GC38-0210.

NAME

rm, rmdir - remove files or directories

SYNOPSIS

rm [**-fri**] file ...

rmdir dir ...

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 a file has no write permission and the standard input is a terminal, its 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. No questions are asked when the **-f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **-i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **-r**, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2).

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file .. merely to avoid the antisocial consequences of inadvertently doing something like:

rm -r .*

It is forbidden to remove a file which is being executed, e.g.,

/bin/rm /bin/rm

The error message given in this case is

/bin/rm: nnn **mode**

where *nnn* is the file access mode of the file.

NAME

rmdel - remove a delta from an SCCS file

SYNOPSIS

rmdel -rSID files

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the *SID* specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get*(1)) exists for the named SCCS file, the *SID* specified must *not* appear in the *p-file*).

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

x-file (see *delta*(1))
z-file (see *delta*(1))

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

rmount, rumount - mount and dismount remote file system

SYNOPSIS

/etc/rmount [rdirectory nodename directory [**-r**]]

/etc/rumount rdirectory nodename

DESCRIPTION

Rmount announces to the system that a remote removable file system directory (*rdirectory*) is present on the system *nodename*. *Rdirectory* and *directory* must exist already; *directory* becomes the initial path name of the remote directory.

These commands, along with *mount*(1M) and *umount*(1M), maintain a table of remote and locally mounted devices. If invoked with no arguments, *rmount* prints the table. This table has two parts: the first part lists all locally mounted file systems plus remote directories mounted on local directories; the second lists all remote directories on which local directories are mounted.

The optional last argument ("-r") indicates that the file is to be mounted read-only.

Rumount announces to the system that the removable file system directory previously mounted at *rdirectory* on *nodename* is to be removed.

FILES

/etc/mnttab rmount table

SEE ALSO

mount(1M), node(1M), mount(2), rmount(2), mnttab(5).

DIAGNOSTICS

Rumount complains if the remote file system directory is not mounted or if it is busy. The remote file system directory is busy if it contains a locally opened file or is some local user's working directory.

NOTES

This command is available on the Plexus Network Operating System (NOS) only.

NAME

rsh - restricted shell (command interpreter)

SYNOPSIS

rsh [flags] [name [arg1 ...]]

DESCRIPTION

Rsh is a restricted version of the standard command interpreter *sh*(1). 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 the following are disallowed:

cd
setting the value of **$PATH**
command names containing /
> and >>

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*(1).

The system administrator often sets up a directory of commands that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

SEE ALSO

sh(1), profile(5).

NAME

runacct - run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [mmdd [state]]

DESCRIPTION

Runacct is the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *Runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes.

Runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail (see *mail*(1)) is sent to **root** and **adm**, and *runacct* terminates. *Runacct* uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

Runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *Runacct* then looks in **statefile** to see what it has done and to determine what to process next. *States* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee* into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Any installation-dependent accounting programs can be included here.
CLEANUP	Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*.

```
nohup runacct 2> /usr/adm/acct/nite/fd2log &
```

To restart *runacct*.

```
nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &
```

To restart *runacct* at a specific *state*.

```
nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &
```

FILES

/usr/lib/acct/runacct
/usr/adm/wtmp
/usr/adm/pacct[1-9]
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct[1-9].*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), acct(2), acct(5), utmp(5).
The UNIX Accounting System by H. S. McCreary.

DIAGNOSTICS

Self explanatory.

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state,* remove the last **ptacct** file because it will not be complete.

NAME

sact - print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **-e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (i.e. executed a *get* for editing).
Field 4	contains the date that **get -e** was executed.
Field 5	contains the time that **get -e** was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sag - system activity graph

SYNOPSIS

sag [**-s** time] [**-e** time] [**-T** term] [**-uirwcohdpaf**] [file]

DESCRIPTION

Sag displays, in a graphical form, the system activity of the UNIX operating system during a specified time interval. *File* is the file that contains the daily system activity information, default is **/usr/adm/sa/sa***dd*, where *dd* is today's day of the month. *Sag* has the following options:

- **-s** *time* Begin graph at *time* specified as *hh:mm*. Default is **08:00**.
- **-e** *time* End graph at *time* specified as *hh:mm*. Default is **18:00**.
- **-T** *term* Translate output to a form suitable for terminal *term*. If this option is not used, the environment variable **$TERM** (see *environ*(7)) is used. Refer to *tplot*(1G) for available types of terminals.
- **-u** Plot CPU utilization, showing proportion of user, system and idle time (default option).
- **-i** Plot percent of time the CPU was idle and waiting on block I/O, waiting on swap in or swap out, or waiting on physical I/O.
- **-r** Plot logical reads/minute and block reads/minute.
- **-w** Plot logical writes/minute and block writes/minute.
- **-c** Plot buffer cache hit ratios for reads and for writes.
- **-o** Plot block transfer rate between system buffers and devices, showing reads/minute, writes/minute, and the sum of reads and writes/minute.
- **-h** Plot bytes read/second by system call *read*(2) and bytes written/second by system call *write*(2).
- **-d** Plot the sum of reads and writes/minute for each of the first three disk drives.
- **-p** Plot process switches/second, process preemptions/second and system calls/second.
- **-a** Plot process swapins/minute and process swapouts/minute.
- **-f** Plot file access activities: iget/second, namei/second, and directory blocks read/second.

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

graph(1G), tplot(1G), sar(8).

NOTES

Plotted data points are extracted from the system activity file, **/usr/adm/sa/sa***dd*, which is written under the control of *cron*(1M), normally every 20 minutes between 8:00 and 18:00 on weekdays, and hourly at other times.

In the event of a system outage, the system activity counters are reset to zero when the system is rebooted. This discontinuity is shown by a gap in the plotted data.

DIAGNOSTICS

"terminal type not known" if **$TERM** is not set and the **-T** option is not specified.

NAME

scc - C compiler for stand-alone programs

SYNOPSIS

scc [+[lib]] [option] ... [file] ...

DESCRIPTION

Scc prepares the named files for stand-alone execution. The *option* and *file* arguments may be anything that can legally be used with the *cc* command; it should be noted, though, that the **-p** (profiling) option, as well as any object module that contains system calls, will cause the executable not to run.

Scc defines the compiler constant, **STANDALONE**, so that sections of C programs may be compiled conditionally for when the executable will be run stand-alone.

The first argument specifies an auxiliary library that defines the device configuration of the computer for which the stand-alone executable is being prepared. If no +*lib* argument is specified, +**A** is assumed. If the + argument is specified alone, no configuration library is loaded unless the user supplies his own.

Scc gets startup code from **/lib/crt2.0**.

FILES

/lib/crt2.0 execution start-off
/usr/lib/lib2A.a stand-alone library

SEE ALSO

cc(1), ld(1), a.out(5).
A Stand-alone Input/Output Library, by S. R. Eisen.

NAME

sccsdiff - compare two versions of an SCCS file

SYNOPSIS

sccsdiff -rSID1 **-r**SID2 **[-p] [-s**n] files

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

-r*SID?* *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.

-p pipe output for each file through *pr*(1).

-s*n* *n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

"*file*: No differences" If the two versions are the same.
Use *help*(1) for explanations.

NAME

script - make typescript of terminal session

SYNOPSIS

/usr/plx/script [**-n**] [**-s**] [**-a**] [**-q**] [**-S** shell] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is saved in a file, and can be sent to the line printer later with *lpr.* If a file name is given, the typescript is saved there. If not, the typescript is saved in the file *typescript.*

To exit script, type control D. This sends an end of file to all processes you have started up, and causes script to exit. For this reason, control D behaves as though you had typed an infinite number of control D's.

This program is useful when using a CRT and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a CRT when hard-copy terminals are in short supply.

The options control what shell is used. **-n** asks for the new shell, **-s** asks for the standard shell. **-S** lets you specify any shell you want. The default depends on the system: /bin/csh is used where possible, otherwise /bin/sh. If the requested shell is not available, *script* uses any shell it can find.

The **-q** flag asks for "quiet mode", where the "script started" and "script done" messages are turned off. The **-a** flag causes script to append to the typescript file instead of creating a new file.

NOTES

This command is based on a similar one from the University of California at Berkeley.

BUGS

Since UNIX has no way to write an end-of-file down a pipe without closing the pipe, there is no way to simulate a single control D without ending script.

The new shell has its standard input coming from a pipe rather than a tty, so stty will not work, and neither will ttyname.

When the user interrupts a printing process, *script* attempts to flush the output backed up in the pipe for better response. Usually the next prompt also gets flushed.

NAME

sdiff - side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

Sdiff uses the output of *diff*(1) 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* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

-l Only print the left side of any lines that are identical.

-s Do not print identical lines.

-o *output* Use 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*(1), 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	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed - stream editor

SYNOPSIS

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.

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*(1) modified thus:

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 new-line *embedded* in the pattern space.

A period **.** matches any character except the *terminal* new-line 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 non-selected 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 **** to hide the new-line. 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 Append. Place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the **:** command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) **c**
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**
text Insert. Place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) **p** Print. Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

- **g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
- **p** Print the pattern space if a replacement was made.
- **w** *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfile* Write. Append the pattern space to *wfile*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function*
Don't. Apply 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) **=** Place the current line number on the standard output as a line.

(2) **{** Execute the following commands through a matching **}** only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO

awk(1), ed(1), grep(1).
SED-A Non-interactive Text Editor by L. E. McMahon.

NAME

send, gath - gather files and/or submit RJE jobs

SYNOPSIS

gath [**-ih**] file . . .

send argument . . .

DESCRIPTION

Gath

Gath concatenates the named files and writes them to the standard output. Tabs are expanded into spaces according to the format specification for each file (see *fspec*(5)). The size limit and margin parameters of a format specification are also respected. Non-graphic characters other than tabs are identified by a diagnostic message and excised. The output of *gath* contains no tabs unless the **-h** flag is set, in which case the output is written with standard tabs (every eighth column).

Any line of any of the files which begins with ~ is interpreted by *gath* as a control line. A line beginning "~ " (tilde,space) specifies a sequence of files to be included at that point. A line beginning **~!** specifies a UNIX command; that command is executed, and its output replaces the **~!** line in the *gath* output.

Setting the **-i** flag prevents control lines from being interpreted and causes them to be output literally.

A file name of - at any point refers to standard input, and a control line consisting of **~.** is a logical EOF. Keywords may be defined by specifying a replacement string which is to be substituted for each occurrence of the keyword. Input may be collected directly from the terminal, with several alternatives for prompting. In fact, all of the special arguments and flags recognized by the *send* command are also recognized and treated identically by *gath*. Several of them only make sense in the context of submitting an RJE job.

Send

Send is a command-level interface to the RJE subsystems. It allows the user to collect input from various sources in order to create a run stream consisting of card images, and submit this run stream for transmission to a host computer.

Possible sources of input to *send* are: ordinary files, standard input, the terminal, and the output of a command or shell file. Each source of input is treated as a virtual file, and no distinction is made based upon its origin. Typical input is an ASCII text file of the sort that is created by the editor *ed*(1). An optional format specification appearing in the first line of a file (see *fspec*(5)) determines the settings according to which tabs are expanded into spaces. In addition, lines that begin with ~ are normally interpreted as commands controlling the execution of *send*. They may be used to set or reset flags, to define keyword substitutions, and to open new sources of input in the midst of the current source. Other text lines are translated one-for-one into card images of the run stream.

The run stream that results from this collection is treated as one job by the RJE subsystems. *Send* prints the card count of the run stream, and the queuer that is invoked prints the name of the temporary file that holds the job while it is awaiting transmission. The initial card of a job submitted to an IBM host must have a // in the first column. The initial card of a job submitted to a UNIVAC host must begin with a "@RUN" or "\ run", etc. Any cards preceding these will be excised. If a host computer is not specified before the first card of the runstream is ready to be sent, *send* will select a reasonable default. In the case of an IBM job, all cards beginning with **/*$** will be excised from the runstream, because they are HASP command cards.

The arguments that *send* accepts are described below. An argument is interpreted according to the first pattern that it matches. Preceding a character with **** causes it to loose any special

meaning it might otherwise have when matching against an argument pattern.

.	Close the current source.
-	Open standard input as a new source.
+	Open the terminal as a new source.
:spec:	Establish a default format specification for included sources, e.g., **:m6t-12:**
:message	Print message on the terminal.
-:prompt	Open standard input and, if it is a terminal, print *prompt*.
+:prompt	Open the terminal and print *prompt*.
-flags	Set the specified flags, which are described below.
+flags	Reset the specified flags.
=flags	Restore the specified flags to their state at the previous level.
!command	Execute the specified UNIX *command* via the one-line shell, with input redirected to **/dev/null** as a default. Open the standard output of the command as a new source.
$line	Collect contiguous arguments of this form and write them as consecutive lines to a temporary file; then have the file executed by the shell. Open the standard output of the shell as a new source.
@directory	The current directory for the send process is changed to *directory*. The original directory will be restored at the end of the current source.
~comment	Ignore this argument.
?:keyword	Prompt for a definition of *keyword* from the terminal unless *keyword* has an existing definition.
?keyword=xx	Define the *keyword* as a two digit hexadecimal character code unless it already has a non null replacement.
?keyword=string	Define the *keyword* in terms of a replacement string unless it already has a non null replacement.
=:keyword	Prompt for a definition of *keyword* from the terminal.
keyword=xx	Define *keyword* as a two-digit hexadecimal character code.
keyword=string	Define *keyword* in terms of a replacement string.
host	The host machine that the job should be submitted to. It can be any name that corresponds to one in the first column of the RJE configuration file (**/usr/rje/lines**).
file-name	Open the specified file as a new source of input.

When commands are executed via **$** or **!** the shell environment (see *environ*(7)) will contain the values of all send keywords that begin with **$** and have the syntax of a shell variable.

The flags recognized by *send* are described in terms of the special processing that occurs when they are set:

-l List card images on standard output. EBCDIC characters are translated back to ASCII.

-q Do not output card images.

-f Do not fold lower case to upper.

-t Trace progress on diagnostic output, by announcing the opening of input sources.

-k Ignore the keywords that are active at the previous level and erase any keyword definitions that have been made at the current level.

-r Process included sources in raw mode; pack arbitrary 8-bit bytes one per column (80 columns per card) until an EOF.

-i Do not interpret control lines in included sources; treat them as text.

-s Make keyword substitutions before detecting and interpreting control lines.

-y Suppress error diagnostics and submit job anyway.

-g Gather mode, qualifying **-I** flag; list text lines before converting them to card images.

-h Write listing with standard tabs.

-p Prompt with * when taking input from the terminal.

-m When input returns to the terminal from a lower level, repeat the prompt, if any.

-a Make **-k** flag propagate to included sources, thereby protecting them from keyword substitutions.

-c List control lines on diagnostic output.

-d Extend the current set of keyword definitions by adding those active at the end of included sources.

-x This flag guarantees that the job will be transmitted in the order of submission (relative to other jobs sent with this flag).

Control lines are input lines that begin with ~. In the default mode **-ir**, they are interpreted as commands to *send*. Normally they are detected immediately and read literally. The **-s** flag forces keyword substitutions to be made before control lines are intercepted and interpreted. This can lead to unexpected results if a control line uses a keyword which is defined within an immediately preceding **~$** sequence. Arguments appearing in control lines are handled exactly like the command arguments to *send*, except that they are processed at a nested level of input.

The two possible formats for a control line are: "~argument" and "~ argument ...". In the first case, where the ~ is not followed by a space, the remainder of the line is taken as a single argument to *send*. In the second case, the line is parsed to obtain a sequence of arguments delimited by spaces. In this case the quotes ' and " may be employed to pass embedded spaces.

The interpretation of the argument . is chosen so that an input line consisting of ~. is treated as a logical EOF. The following example illustrates some of the above conventions:

```
send -
~ argument ...
~.
```

This sequence of three lines is equivalent to the command synopsis at the beginning of this description. In fact, the - is not even required. By convention, the *send* command reads standard input if no other input source is specified. *Send* may therefore be employed as a filter with side-effects.

The execution of the *send* command is controlled at each instant by a current environment, which includes the format specification for the input source, a default format specification for included sources, the settings of the mode flags, and the active set of keyword

definitions. This environment can be altered dynamically. When a control line opens a new source of input, the current environment is pushed onto a stack, to be restored when input resumes from the old source. The initial format specification for the new source is taken from the first line of the file. If none is provided, the established default is used or, in its absence, standard tabs. The initial mode settings and active keywords are copied from the old environment. Changes made while processing the new source will not affect the environment of the old source, with one exception: if **-d** mode is set in the old environment, the old keyword context will be augmented by those definitions that are active at the end of the new source.

When *send* first begins execution, all mode flags are reset, and the values of the shell environment variables become the initial values for keywords of the same name with a **$** prefixed.

The initial reset state for all mode flags is the **+** state. In general, special processing associated with a mode *N* is invoked by flag *-N* and is revoked by flag *+N*. Most mode settings have an immediate effect on the processing of the current source. Exceptions to this are the **-r** and **-i** flags, which apply only to included source, causing it to be processed in an uninterpreted manner.

A keyword is an arbitrary 8-bit ASCII string for which a replacement has been defined. The replacement may be another string, or (for IBM RJE only) the hexadecimal code for a single 8-bit byte. At any instant, a given set of keyword definitions is active. Input text lines are scanned, in one pass from left to right, and longest matches are attempted between substrings of the line and the active set of keywords. Characters that do not match are output, subject to folding and the standard translation. Keywords are replaced by the specified hexadecimal code or replacement string, which is then output character by character. The expansion of tabs and length checking, according to the format specification of an input source, are delayed until substitutions have been made in a line.

All of the keywords definitions made in the current source may be deleted by setting the **-k** flag. It then becomes possible to reuse them. Setting the **-k** flag also causes keyword definitions active at the previous source level to be ignored. Setting the **+k** flag causes keywords at the previous level to be ignored but does not delete the definitions made at the current level. The **=k** argument reactivates the definitions of the previous level.

When keywords are redefined, the previous definition at the same level of source input is lost, however the definition at the previous level is only hidden, to be reactivated upon return to that level unless a **-d** flag causes the current definition to be retained.

Conditional prompts for keywords, **?:A**,/p which have already been defined at some higher level to be null or have a replacement will simply cause the definitions to be copied down to the current level; new definitions will not be solicited.

Keyword substitution is an elementary macro facility that is easily explained and that appears useful enough to warrant its inclusion in the *send* command. More complex replacements are the function of a general macro processor (*m4*(1), perhaps). To reduce the overhead of string comparison, it is recommended that keywords be chosen so that their initial characters are unusual. For example, let them all be upper case.

Send performs two types of error checking on input text lines. Firstly, only ASCII graphics and tabs are permitted in input text. Secondly, the length of a text line, after substitutions have been made, may not exceed 80 bytes for IBM, or 132 bytes for UNIVAC. The length of each line may be additionally constrained by a size parameter in the format specification for an input source. Diagnostic output provides the location of each erroneous line, by line number and input source, a description of the error, and the card image that results. Other routine errors that are announced are the inability to open or write files, and abnormal exits

from the shell. Normally, the occurrence of any error causes *send*, before invoking the queuer, to prompt for positive affirmation that the suspect run stream should be submitted.

For IBM hosts, *send* is required to translate 8-bit ASCII characters into their EBCDIC equivalents. The conversion for 8-bit ASCII characters in the octal range 040-176 is based on the character set described in "Appendix H" of *IBM System/370 Principles of Operation* (IBM SRL GA22-7000). Each 8-bit ASCII character in the range 040-377 possesses an EBCDIC equivalent into which it is mapped, with five exceptions: ~ into ¬, 0345 into ~, 0325 into ¢, 0313 into |, 0177 (DEL) is illegal. In listings requested from *send* and in printed output returned by the subsystem, the reverse translation is made with the qualification that EBCDIC characters that do not have valid 8-bit ASCII equivalents are translated into ^. UNIVAC hosts, on the other hand, operate in ASCII code, and any translations between ASCII and field-data are made, in accordance with the UNIVAC standard, by the host computer.

Additional control over the translation process is afforded by the **-f** flag and hexadecimal character codes. As a default, *send* folds lower-case letters into upper case. For UNIVAC RJE it does more: the entire ASCII range 0140-0176 is folded into 0100-0136, so that `, for example, becomes @. In either case, setting the **-f** flag inhibits any folding. Non-standard character codes are obtained as a special case of keyword substitution.

SEE ALSO

m4(1), orjestat(1C), rjestat(1C), sh(1), fspec(5), ascii(7), hasp(8), rje(8), uvac(8).
Guide to IBM Remote Job Entry for PWB/UNIX Users by A. L. Sabsevitz and E. J. Finger.
UNIX Remote Job Entry User's Guide by K. A. Kelleman.

BUGS

Standard input is read in blocks, and unused bytes are returned via *lseek*(2). If standard input is a pipe, multiple arguments of the form - and -:*prompt* should not be used, nor should the logical EOF (~.).

NAME

setmnt - establish mnttab table

SYNOPSIS

/etc/setmnt

DESCRIPTION

Setmnt creates the **/etc/mnttab** table (see *mnttab*(5)), which is needed for the *mount*(1M), *rmount*(1M), *umount*(1M), and *rumount*(1M) commands. *Setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

filesys node [dir]

where *filesys* is the name of the file system's *special file* (e.g., "dk1") or the remote directory and *node* is the root name of that file system or the remote node. The third parameter, *dir*, is only used to generate *rmount* entries. It is the local directory name. Thus *filesys* and *node* become the first two strings in the *mnttab*(5) entry.

FILES

/etc/mnttab

SEE ALSO

mnttab(5).

BUGS

Evil things will happen if *filesys* or *node* are longer than 50 characters.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

sh - shell, the standard command programming language

SYNOPSIS

sh [**-ceiknrstuvx**] [args]

DESCRIPTION

Sh is a 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 non-blank *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*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) 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*(2) 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 (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

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 *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 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**
: 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 sub-shell.

{*list***;}**
: *list* is simply executed.

The following 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 { }

Comments.

A word beginning with **#** causes that word and all the following characters up to a new-line 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 new-lines are removed.

Parameter Substitution.

The character **$** is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*.

${*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 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 the message "parameter null or not set" is printed.

${*parameter***:+***word***}**

If *parameter* is set and is non-null then 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 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 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:

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.

PS1 Primary prompt string, by default "**$** ".
PS2 Secondary prompt string, by default "**>** ".
IFS Internal field separators, normally **space**, **tab**, and **new-line**.

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*(1)).

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.

File Name Generation.
Following substitution, each command *word* is scanned for the characters *, **?**, and **[**. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

***** 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.

Quoting.
The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | < > new-line space tab

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 ('' ''), except a single quote, are quoted. Inside double quote 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 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.

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; 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 then it is created; otherwise, it is truncated to zero length.
>>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 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, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\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*(2)). 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, then 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 **&** 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*(7)) 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 affects 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 thus composed of any unmodified name-value pairs originally inherited by the shell, 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. The following first prints **a=b c** and then **c**:

```
echo a=b c
set -k
echo a=b c
```

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 (but see also the **trap** command below).

When the shell associated with a terminal terminates, processes spun off with **&** are sent the SIGHUP signal. See *exit*(2).

Execution.

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec*(2).

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 path name, 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 sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

Special Commands.

The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*.

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 then 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, then a list of all names that are exported in this shell is printed.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* ....

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, then a list of all *readonly* names is printed.

set [**-ekntuvx** [*arg* ...]]

- **-e** If the shell is non-interactive then exit immediately if a command exits with a non-zero exit status.
- **-k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **-n** Read commands but do not execute them.
- **-t** Exit after reading and executing one command.
- **-u** Treat unset variables as an error when substituting.
- **-v** Print shell input lines as they are read.
- **-x** Print commands and their arguments as they are executed.
- **--** Do 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 then the values of all names are printed.

shift

The positional parameters from **$2** ... are renamed **$1** ....

test

Evaluate conditional expressions. See *test*(1) for usage and description.

times

Print 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. 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 then all trap(s) *n* 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 *n* is 0 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.

umask [*nnn*]

The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait Wait for all child processes to terminate report the termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is always zero.

Invocation.

If the shell is invoked through *exec*(2) and the first character of argument zero 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 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. Any remaining arguments specify the positional parameters. Shell output 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, then 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*(1)).

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 non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), env(1), login(1), newgrp(1), rsh(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(5), profile(5), environ(7).

BUGS

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.

SIZE(1)

NAME

size - size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. If no file is specified, **a.out** is used.

SEE ALSO

a.out(5).

NAME

sleep - suspend execution for an interval

SYNOPSIS

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(2), sleep(3C).

BUGS

Time must be less than 65536 seconds.

NAME

sno - SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

Sno is a SNOBOL compiler and interpreter (with slight differences). *Sno* obtains input from the concatenation of the named *files* and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspit**.

Sno differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

```
a ** b          unanchored search for b.
a *x* b = x c   unanchored assignment
```

There is no back referencing.

```
x = "abc"
a *x* x         is an unanchored search for abc.
```

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and * must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable **sysppt** is not available.

SEE ALSO

awk(1).

"SNOBOL, a String Manipulation Language," by D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* **11** (1964), pp. 21-30.

NAME

sort - sort and/or merge files

SYNOPSIS

sort [**-cmubdfinrtx**] [+pos1 [-pos2]] ... [**-o** output] [names]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name - means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d "Dictionary" order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b**.

r Reverse the sense of comparisons.

t*x* "Tab character" separating fields is *x*.

The notation +*pos1* -*pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first non-blank in the field; **b** is attached independently to *pos2*. A missing *.n* means .0; a missing -*pos2* means the end of the line. Under the -**t***x* option, fields are strings separated by *x*; otherwise fields are non-empty non-blank strings separated by blanks.

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.

These option arguments are also understood:

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 Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

o The next argument 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.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words (capitalized words differ from uncapitalized):

```
sort -u +0f +0 list
```

Print the password file (*passwd*(5)) sorted by user ID (the third colon-separated field):

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month-day) entries (the options **-um** with just one input file make the choice of a unique representative from a set of

equal lines predictable):

```
sort -um +0 -1 dates
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions and for disorder discovered under option **-c**.

BUGS

If you try to sort lines longer than 2048 characters, the lines may be truncated. If lines are truncated, a warning message appears on standard error.

NAME

spell, spellin, spellout - find spelling errors

SYNOPSIS

spell [options] [files]

/usr/lib/spell/spellin [list]

/usr/lib/spell/spellout [**-d**] 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*(1), *tbl*(1), and *eqn*(1) 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*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the **-x** option, every plausible stem is printed with = for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, 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. 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.

Two routines help maintain the hash lists used by *spell* (both expect a list of words, one per line, from the standard input): *spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word read from the standard input, and prints on the standard output those that are missing from (or, with the **-d** option, present in) the hash list.

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
/tmp/spell.$$	temporary
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1), typo(1).

BUGS

The spelling list's 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 dictionary that is added to the hashed *list* via *spellin*.
British spelling was done by an American.

NAME

spline - interpolate smooth curve

SYNOPSIS

spline [options]

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 (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

The following *options* are recognized, each as a separate argument:

-a Supply 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:
$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$
is set by the next argument (default $k = 0$).

-n *n* Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).

-p Make output periodic, i.e., match 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).

SEE ALSO

graph(1G).

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

NAME

split - split a file into pieces

SYNOPSIS

split [*-n*] [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, 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 **-** is given in its stead, then the standard input file is used.

If *n* is less than 1, *split* sets *n* to 1000.

SEE ALSO

bfs(1), csplit(1).

NAME

st - synchronous terminal control

SYNOPSIS

/etc/stload
/etc/stcntrl [on | off]

DESCRIPTION

The *stload* command file is used to load the synchronous terminal prototype script, **/etc/proto**, into the designated KMC11-B microprocessor, and start execution of the script. As supplied, *stload* uses **/dev/kmc0**; it may need local modification if another KMC11-B is being used.

The *stcntrl* command is used to activate and deactivate the synchronous terminal driver.

The **/etc/rc** file should contain the following multi-user entries:

```
/etc/stload
/etc/stcntrl on
```

while **/etc/shutdown** should have:

```
/etc/stcntrl off
```

FILES

/etc/stproto	synchronous terminal prototype script
/dev/kmc?	KMC11-B microprocessor
/dev/vpm?	virtual protocol machine
/dev/st0	synchronous terminal control channel
/dev/st?	synchronous terminal user channels

SEE ALSO

kmc(4), st(4), trace(4), vpm(4).

BUGS

The **stcntrl.c** file assumes that **/dev/vpm0** is the *vpm* device being used for the first (and usually only) synchronous terminal controller. If some other *vpm* device is being used, the **stcntrl.c** file must be modified and rebuilt.

NAME

strings - find the printable strings in a object, or other binary, file

SYNOPSIS

/usr/plx/strings [**-**] [**-o**] [*-number*] file ...

DESCRIPTION

Strings looks for ASCII strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. 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 offset in the file (in octal). 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.

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

od(1)

BUGS

The algorithm for identifying strings is extremely primitive.

STRIP(1)

NAME

strip - remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **-s** option of *ld*.

If *name* is an archive file, *strip* will remove the local symbols from any *a.out* format files it finds in the archive. Certain libraries, such as those residing in **/lib**, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

FILES

/tmp/stm* temporary file

SEE ALSO

ld(1).

NAME

stty - set the options for a terminal

SYNOPSIS

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, it reports all of the option settings; with the **-g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *tty*(4). Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (-parenb) enable (disable) parity generation and detection.

parodd (-parodd) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *tty*(4)).

0 hang up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible.

hupcl (-hupcl) hang up (do not hang up) DATA-PHONE® connection on last close.

hup (-hup) same as **hupcl (-hupcl)**.

cstopb (-cstopb) use two (one) stop bits per character.

cread (-cread) enable (disable) the receiver.

clocal (-clocal) assume a line without (with) modem control.

Input Modes

ignbrk (-ignbrk) ignore (do not ignore) break on input.

brkint (-brkint) signal (do not signal) INTR on break.

ignpar (-ignpar) ignore (do not ignore) parity errors.

parmrk (-parmrk) mark (do not mark) parity errors (see *tty*(4)).

inpck (-inpck) enable (disable) input parity checking.

istrip (-istrip) strip (do not strip) input characters to seven bits.

inlcr (-inlcr) map (do not map) NL to CR on input.

igncr (-igncr) ignore (do not ignore) CR on input.

icrnl (-icrnl) map (do not map) CR to NL on input.

iuclc (-iuclc) map (do not map) upper-case alphabetics to lower case on input.

ixon (-ixon) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (-ixany) allow any character (only DC1) to restart output.

ixoff (-ixoff) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (-opost) post-process output (do not post-process output; ignore all other output modes).

olcuc (-olcuc) map (do not map) lower-case alphabetics to upper case on output.

onlcr (-onlcr) map (do not map) NL to CR-NL on output.

ocrnl (-ocrnl) map (do not map) CR to NL on output.

onocr (-onocr) do not (do) output CRs at column zero.

onlret (-onlret) on the terminal NL performs (does not perform) the CR function.

ofill (-ofill) use fill characters (use timing) for delays.

ofdel (-ofdel) fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 select style of delay for carriage returns (see *tty*(4)).
nl0 nl1 select style of delay for line-feeds (see *tty*(4)).
tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *tty*(4)).
bs0 bs1 select style of delay for backspaces (see *tty*(4)).
ff0 ff1 select style of delay for form-feeds (see *tty*(4)).
vt0 vt1 select style of delay for vertical tabs (see *tty*(4)).

Local Modes

isig (-isig) enable (disable) the checking of characters against the special control characters INTR and QUIT.
icanon (-icanon) enable (disable) canonical input (ERASE and KILL processing).
xcase (-xcase) canonical (unprocessed) upper/lower-case presentation.
echo (-echo) echo back (do not echo back) every character typed.
echoe (-echoe) echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed 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) echo (do not echo) NL after KILL character.
lfkc (-lfkc) the same as **echok (-echok)**; obsolete.
echonl (-echonl) echo (do not echo) NL.
noflsh (-noflsh) disable (enable) flush after INTR or QUIT.

Control Assignments

control-character c set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **min**, or **time** (**min** and **time** are used with **-icanon**; see *tty*(4)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "**^d**" is a **CTRL-d**); "**^?**" is interpreted as DEL and "**^-**" is interpreted as undefined.
line *i* set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity** enable **parenb** and **cs7**.
oddp enable **parenb**, **cs7**, and **parodd**.
-parity, **-evenp**, or **-oddp** disable **parenb**, and set **cs8**.
raw (**-raw** or **cooked**) enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).
nl (-nl) unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (-lcase) set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (-LCASE) same as **lcase (-lcase)**.
tabs (-tabs or **tab3)** preserve (expand to spaces) tabs when printing.
ek reset ERASE and KILL characters back to normal **#** and **@**.
sane resets all modes to some reasonable values.
term set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

NOTES

Plexus supports the following additional switches when the ICP is configured to accept the CTS signal:

icts enable clear_to_send (CTS). Note that when **icts** is enabled, XON/XOFF is still enabled.

ictslo	inverts the polarity of the **icts**.
flushi	flushes the input queue only.
flusho	flushes the output queue only.
flush	flushes both input and output queues.
xon	restarts suspended output.
xoff	suspends output.
break	sends a break (zero bits for 0.25 second)

SEE ALSO

tabs(1), ioctl(2), tty(4).

NAME

style - analyze surface characteristics of a document

SYNOPSIS

/usr/plx/style [**-ml**] [**-mm**] [**-a**] [**-e**] [**-l** num] [**-r** num] [**-p**] [**-P**] file ...

DESCRIPTION

Style analyzes the surface characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because *style* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **-ms** may be overridden with the flag **-mm**. The flag **-ml**, which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The other options are used to locate sentences with certain characteristics.

-a print all sentences with their length and readability index.

-e print all sentences that begin with an expletive.

-p print all sentences that contain a passive verb.

-l*num* print all sentences longer than *num*.

-r*num* print all sentences whose readability index is greater than *num*.

-P print parts of speech of the words in the document.

SEE ALSO

deroff(1), diction(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

NAME

su - become super-user or another user

SYNOPSIS

su [**-**] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already super-user). If the password is correct, *su* will execute a new shell with the user ID set to that of the specified user. To restore normal user ID privileges, type an **EOF** to the new shell.

Any additional arguments are passed to the shell, permitting the super-user to run shell procedures with restricted privileges (an *arg* of the form **-c** *string* executes *string* via the shell). When additional arguments are passed, **/bin/sh** is always used. When no additional arguments are passed, *su* uses the shell specified in the password file.

An initial **-** flag causes the environment to be changed to the one that would be expected if the user actually logged in again. This is done by invoking the shell with an *arg0* of **-su** causing the **.profile** in the home directory of the new user ID to be executed. Otherwise, the environment (up to 30 environment variables) is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for root. Note that the **.profile** can check *arg0* for **-sh** or **-su** to determine how it was invoked.

FILES

/etc/passwd	system's password file
$HOME/.profile	user's profile

SEE ALSO

env(1), login(1), sh(1), environ(7).

NAME

sum - sum and count blocks in a file

SYNOPSIS

sum [**-r**] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum. The alternate algorithm is compatible with the Version 7 *sum*.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

sync - update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. See *sync*(2) for details.

SEE ALSO

update(1), sync(2).

NAME

tabs - set tabs on a terminal

SYNOPSIS

tabs [tabspec] [+**m**n] [-Ttype]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **-8**, i.e., UNIX "standard" 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* Gives the name of one of a set of "canned" 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:

<: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:

<: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

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 UNIX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff*(1) **-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 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 tab lists 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. 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*(1) command:

```
tabs -- file; pr file
```

Any of the following may be used also; 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*(7). If no **-T** flag is supplied, *tabs* searches for the **$TERM** value in the *environment* (see *environ*(7)). If no *type* can be found, *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.

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

nroff(1), environ(7), term(7).

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.

NAME

tail - deliver the last part of a file

SYNOPSIS

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 fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1).

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

tape - tape manipulation

SYNOPSIS

tape [-9ctf [filename] **]** [command [options]]

DESCRIPTION

Tape executes special commands on 9-track and cartridge tapes.

Tape accesses the filename given via the **-f** switch. If none is given, the program opens the tape filename, **/dev/nrrmh0**. If that fails, it opens **/dev/nrmt0**. The tape file it eventually accesses must be a character special file.

Tape figures out the type of tape (9-track or cartridge) unless the 'c' or '9' switches are given.

The available switches are:

c The tape file is on a cartridge tape. *Tape* does not try figure out the type of tape if this is given.

9 The tape file is on a 9-track tape. *Tape* does not try figure out the type of tape if this is given.

t Print the type of tape.

f *filename* Do the special command on the tape file *filename*. *Filename* must be a character special file.

Tape accepts a *command* after the switches. If none is given, it assumes the command *status* for the 9-track and *srcheof 0* for the cartridge.

These are the acceptable commands and their options for a 9-track tape:

erase *n* Erases a fixed length (approximately 3.5 inches) for each value of n from the current position.

eraseall Erase tape from current position to beyond the end of tape.

status Report the status returned by tape controller.

rforeign Reports the status and block size in bytes of file on tape. Leaves the tape in the middle of the file.

unload Unloads the tape.

space *n* Space forward *n* data blocks.

spaceeof *n* Space forward *n* data blocks. Terminates early if it encounters an end-of-file on tape.

srcheof *n* Position tape to the *n*-th end-of-file mark from the current position. If *n* is omitted, 1 is assumed. *n* should be a positive integer. If *filename* is given with the **f** switch you should specify a tape filename with no rewind, e.g., **/dev/nrmt0**.

rew Rewinds the tape to the load point.

weof Write an end-of-file on tape at current position.

These are the acceptable commands and their options for a cartridge tape:

rew Rewinds the cartridge to the load point.

weof Write an end-of-file on cartridge at current position. This is dangerous because it could overwrite data anywhere on the cartridge. Don't use it if you want the data on the cartridge.

srcheof *n* Position to the *n*-th file on the cartridge. Positioning is always done from the beginning of the cartridge. If *n* is omitted, 1 is assumed. *n* should be a positive integer. If *filename* is given with the **f** switch you should specify a tape filename with no rewind, e.g., **/dev/nrmt0**.

eraseall Erase cartridge. The abbreviation 'erase' works also.

retension Retension the cartridge. This should be done with new cartridges, cartridges that have been unused for some time, or cartridges that produce a hard error when accessing. The abbreviation 'ret' works also. See the *Plexus Cartridge Tape Drive Manual* (Plexus Publication Number 5016) for more information.

The filename **/dev/nrrmh0** is a special device for a 9-track tape. It allows *tape* to do certain *commands* in streaming mode, e.g., *srcheof*. See rm(4).

FILES

/dev/nrmt0
/dev/nrrmh0

SEE ALSO

pt(4), rm(4).

NAME

tar – tape file archiver

SYNOPSIS

tar [key] [files]

DESCRIPTION

Tar saves and restores files on magnetic tape. 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. Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named *files* are written on the end of the tape. The **c** function implies this function.

x The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t The names of the specified files are listed each time that they occur on the tape. If no *files* argument is given, all the names on the tape are listed.

u The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape.

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This command implies the **r** function.

The following characters may be used in addition to the letter that selects the desired function:

0,...,7 This modifier selects the drive on which the tape is mounted. The default is **0**.

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to type 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.

w causes *tar* 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".

f causes *tar* to use the next argument as the name of the archive instead of **/dev/rmt0**. If the name of the file is **–**, *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 tape records. The default is 1, the maximum is 20. The block size is determined automatically when reading tapes (key letters **x** and **t**). Specifying the wrong block size with the **b** option can lead to unpredictable results.

l tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m tells *tar* to not restore the modification times. The modification time of the file will be the time of extraction.

FILES

/dev/rmt0
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.

Complaints if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on file-name length is 100 characters.
The **r** and **u** options do not work.
Tar does not preserve the access modes and ownership of directories. If you need them preserved, use *cpio*(1) instead. See also *find*(1).

NAME

tbl - format tables for nroff or troff

SYNOPSIS

tbl [-TX] [files]

DESCRIPTION

Tbl is a preprocessor that formats tables for *nroff*(1) or *troff*(1). The input files are copied to the standard output, except for lines between **.TS** and **.TE** command lines, which are assumed to describe tables and are re-formatted by *tbl*. (The **.TS** and **.TE** command lines are not altered by *tbl*).

.TS is followed by global options. The available global options are:

center	center the table (default is left-adjust);
expand	make the table as wide as the current line length;
box	enclose the table in a box;
doublebox	enclose the table in a double box;
allbox	enclose each item of the table in a box;
tab (*x*)	use the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi-colon (;).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc. The available key-letters are:

c	center item within the column;
r	right-adjust item within the column;
l	left-adjust item within the column;
n	numerically adjust item in the column: units positions of numbers are aligned vertically;
s	span previous item on the left into this column;
a	center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
^	span down previous entry in this column;
_	replace this entry with a horizontal line;
=	replace this entry with a double horizontal line.

The characters **B** and **I** stand for the bold and italic fonts, respectively; the character **|** indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by **.TE**. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ or =, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only _ or =, then that item is replaced by a single or double line.

Full details of all these and other features of *tbl* are given in the reference manual cited below.

The **-TX** option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments, *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn*(1) or *neqn*(1), *tbl* should come first to minimize the volume of

data passed through pipes.

EXAMPLE

If we let → represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
| | n n .
Household Population
_
Town→Households
→Number→Size
=
Bedminster→789→3.26
Bernards Twp.→3087→3.74
Bernardsville→2018→3.30
Bound Brook→3425→3.04
Bridgewater→7897→3.81
Far Hills→240→3.19
.TE
```

yields:

Household Population		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

TBL-A Program to Format Tables by M. E. Lesk
eqn(1), mm(1), mmt(1), troff(1), mm(7), mv(7).

BUGS

See *BUGS* under *troff*(1).

TC(1)

NAME

tc - phototypesetter simulator

SYNOPSIS

tc [**-t**] [**-s***n*] [**-p***l*] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Wang Laboratories, Inc. C/A/T phototypesetter. The standard output of *tc* is intended for a Tektronix 4014 terminal with ASCII and APL character sets. The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, with overstruck combinations where necessary. Typical usage is:

troff -t files | tc

At the end of each page, *tc* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will *suppress* the screen erase before the next page; **s***n* will cause the next *n* pages to be skipped; and !*cmd* will send *cmd* to the shell.

The command line options are:

-t Don't wait between pages (for directing output into a file).

-s*n* Skip the first *n* pages.

-p*l* Set page length to *l*; *l* may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

SEE ALSO

4014(1), sh(1), tplot(1G), troff(1).

BUGS

Font distinctions are lost.

TEE(1)

NAME

tee - pipe fitting

SYNOPSIS

tee [**-i**] [**-a**] [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.

NAME

test - condition evaluation command

SYNOPSIS

test expr
[expr **]**

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero 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 the string *s1* is non-zero.
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.

NOTES

The *test* facility is embedded into *sh*. Thus it is not available to *csh* or to any other program.

SEE ALSO

find(1), sh(1).

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.

TIME(1)

NAME

time - time a command

SYNOPSIS

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 execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

The times are printed on standard error.

SEE ALSO

timex(1), times(2).

NAME

timex - time a command and generate a system activity report

SYNOPSIS

timex command

DESCRIPTION

The given *command* is executed; after its execution, *timex* prints the elapsed time, the time spent executing *command*, and the time spent in the system, as *time*(1) does. It also reports system activity that occurred during *command* execution, including CPU utilization, I/O activity, system switching and swapping, and file system access. *All* system activity is reported, not just that due to *command*.

The output of *timex* is written on standard error.

SEE ALSO

time(1), sar(8).

NAME

topq - put a print request at the head of the queue

SYNOPSIS

topq id

DESCRIPTION

The *topq* command is for use with the **lp**(1) spooler. It places the request whose identification number is *id* at the top of the print queue, whether or not *lpsched*(1M) is running.

Only super-user can *topq* a request.

FILES

/usr/spool/lp/*

NOTES

Topq is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

lp(1), lphold(1), lprun(1), lpsched(1M).

NAME

touch - update access and modification times of a file

SYNOPSIS

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*(1)) the current time is used. 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(1), utime(2).

NAME

tp - manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or other magnetic tape. 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. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the tape. If files with the same names already exist, they are replaced. "Same" is determined by string comparison, so **./abc** can never be the same as **/usr/sbo/abc** even if **/usr/sbo** is the current directory. If no file argument is given, **.** is the default.

u Updates the tape. **u** is like **r**, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. **u** is the default command if none is given.

d Deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magnetic tapes.

x Extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t Lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

m Specifies magnetic tape as opposed to DECtape.

0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, **x** is default; for magnetic tape **0** is the default.

v Normally *tp* does its work silently. The **v** (verbose) option causes it to type 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.

c Means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u**. This option is assumed with magnetic tape since it is impossible to selectively overwrite magnetic tape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option is known to work only with **x**.

w Causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means "yes", so the file is treated. Null response means "no", and the file does not take part in whatever is being done. Response **x** means "exit"; the *tp* command terminates immediately. In the **x** function, files previously asked about have been extracted already. With **r**, **u**, and **d** no change has been made to the tape.

FILES

/dev/tap?
/dev/mt?

SEE ALSO

ar(1), cpio(1), tar(1).

DIAGNOSTICS

Several; the non-obvious one is "Phase error", which means the file changed after it was selected for dumping but before it was dumped.

BUGS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by *tp* difficult to carry to other machines; *tar*(1) avoids the problem.

Tp does not copy zero-length files to tape.

NAME

tplot - graphics filters

SYNOPSIS

tplot [**-T**terminal [**-e** raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(5)) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** (see *environ*(7)) is used. Known *terminal*s are:

300	DASI 300.
300S	DASI 300s.
450	DASI 450.
4014	Tektronix 4014.
ver	Versatec D1200A. This version of *plot* places a scan-converted image in **/usr/tmp/raster$$** and sends the result directly to the plotter device, rather than to the standard output. The **-e** option causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(5), term(7).

NAME

tr - translate characters

SYNOPSIS

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(1), sh(1), ascii(7).

BUGS

Won't handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

NAME

trmtab - make a new nroff/troff terminal/printer driver table

SYNOPSIS

/usr/src/cmd/term/trmtab tabname

DESCRIPTION

Trmtab makes a new *nroff/troff* driver table and installs it in **/usr/lib/term**. It uses two files, *tab.c* and *code.new*, that must be modified to represent the new terminal/printer.

Tabname is the name of the output file that is installed in **/usr/lib/term**. *Tabname* must begin with "tab".

To use the new driver table, use the "-T" option of *nroff/troff*.

FILES

/usr/src/cmd/term/code.new
/usr/src/cmd/term/tab.c

NOTES

Trmtab is a Plexus command. It is not part of standard SYSTEM III.

SEE ALSO

troff(1)

NAME

troff, nroff - typeset or format text

SYNOPSIS

nroff [options] [files]

troff [options] [files]

DESCRIPTION

Nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers; similarly, *troff* formats text for a Wang Laboratories, Inc., C/A/T phototypesetter. Their capabilities are described in the *NROFF/TROFF User's Manual* cited below.

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

-o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N*-*M* means pages *N* through *M*; an initial -*N* means from the beginning to page *N*; and a final *N*- means from *N* to the end. (See *BUGS* below.)

-n*N* Number first generated page *N*.

-s*N* Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). This option does not work if the output of *nroff* is piped through *col*(1). *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed. When *nroff* (*troff*) halts between pages, an ASCII **BEL** (in *troff*, the message **page stop**) is sent to the terminal.

-ra*N* Set register *a* (which must have a one-character name) to *N*.

-i Read standard input after *files* are exhausted.

-q Invoke the simultaneous input-output mode of the **.rd** request.

-z Print only messages generated by **.tm** (terminal message) requests.

-m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

-c*name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

-k*name* Compact the macros used in this invocation of *nroff*/*troff*, placing the output in files **[dt].***name* in the current directory (see the May 1979 Addendum to the *NROFF/TROFF User's Manual* for details of compacting macro files).

Nroff only:

-T*name* Prepare output for specified terminal. Known *name*s are **37** for the (default) TELETYPE® Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, and **2631** for the Hewlett Packard 2631 line printer.

-e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.

-h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

-u*n* Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

Troff only:

-t Direct output to the standard output instead of the phototypesetter.
-f Refrain from feeding out paper and stopping phototypesetter at the end of the run.
-w Wait until phototypesetter is available, if it is currently busy.
-b Report whether the phototypesetter is busy or available. No text processing is done.
-a Send a printable ASCII approximation of the results to the standard output.
-p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
-g Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output (see *gcat*(1C)). This option is not compatible with the **-s** option; furthermore, when this option is invoked, all **.fp** (font position) requests (if any) in the *troff* input must come before the first break, and *no* **.tl** requests may come before the first break.
-T*name* Use font-width tables for device *name* (the font tables are found in **/usr/lib/font/***name*/*). Currently, no *name*s are supported.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*

SEE ALSO

NROFF/TROFF User's Manual by J. F. Ossanna.
A TROFF Tutorial by B. W. Kernighan.
eqn(1), tbl(1), mm(7).
col(1), greek(1), mm(1) (*nroff* only).
gcat(1C), mmt(1), tc(1), mv(7) (*troff* only).

BUGS

Nroff/troff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff/troff* generates may be off by one day from your idea of what the date is.
When *nroff/troff* is used with the **-o***list* option inside a pipeline (e.g., with one or more of *cw*(1), *eqn*(1), and *tbl*(1)), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

true, false - provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tset - set terminal modes

SYNOPSIS

/usr/plx/tset [-] [-hrsIQS] [-e[*c*]] [-E[*c*]]
[-k[*c*]] [-m [*ident*][*test baudrate*]:*type*] [*type*]

DESCRIPTION

Tset causes terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. It is driven by the **/etc/ttytype** and **/etc/termcap** files.

The *type* argument specifies the type of terminal. The type may be any type given in **/etc/termcap.** If *type* is not specified, the terminal type is read from the environment TERM, unless the **-h** flag is set or any **-m** argument was given. In this case the type is read from **/etc/ttytype** (the data base that links port names to terminal types). The port name is determined by a *ttyname*(3) call on the diagnostic output. If the port is not found in **/etc/ttytype,** the terminal type is set to *unknown.*

Ports for which the terminal type is indeterminate are identified in **/etc/ttytype** as *dialup, plugboard,* etc. You can specify how these identifiers should map to an actual terminal type. The mapping flag, **-m,** is followed by the appropriate identifier (a 4 character or longer substring is adequate), an optional test for baud rate, and the terminal type to be used if the mapping conditions are satisfied. If more than one mapping is specified, the first correct mapping prevails. A missing identifier matches all identifiers. Baud rates are specified as with *stty*(1), and are compared with the speed of the diagnostic output. The test may be any combination of: **>**, **=**, **<**, **@**, and **!.** (Note: **@** is a synonym for **=** and **!** inverts the sense of the test. Remember to escape characters meaningful to the shell.)

If the *type* as determined above begins with a question mark, *tset* asks if you really want that type. A null response means to use that type; otherwise, another type can be entered, which is then used instead. (The question mark must be escaped to prevent filename expansion by the shell.)

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ('#' on standard systems), the erase character is changed to a Control-H (backspace). The **-e** flag sets the erase character to be the named character *c* on all terminals, so to override this option you can say **-e#**. The default for *c* is the backspace character on the terminal, usually Control-H. The **-E** flag is identical to **-e** except that it only operates on terminals that can backspace; it might be used with an ASR33. The **-k** option works similarly, with *c* defaulting to Control-X. No kill processing is done if **-k** is not specified. In all of these flags, "^X", where X is any character, is equivalent to control-X.

The **-** option prints the terminal type on the standard output; this can be used to get the terminal type by saying:

```
set termtype = `tset -`
```

If no other options are given, *tset* operates in "fast mode" and *only* outputs the terminal type, bypassing all other processing.

The **-s** outputs *export* and assignment commands (if your default shell is the Bourne shell). Use:

```
tset -s ... > /tmp/tset$$
/tmp/tset$$
rm /tmp/tset$$
```

For the same effect, if you are using *csh*, use:

```
set noglob
set term=(`tset -S ....')
setenv TERM $term[1]
setenv TERMCAP "$term[2]"
unset term
unset noglob
```

The **-S** option only outputs the strings to be placed in the environment variables.

The **-r** option prints the terminal type on the diagnostic output.

The **-Q** option supresses printing the "Erase set to" and "Kill set to" messages.

The **-I** option supresses outputing the terminal initialization strings.

Tset is most useful when included in the **.login** (for *csh*(1)) or **.profile** (for *sh*(1)) 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
```

FILES

/etc/ttytype	Port name to terminal type map database
/etc/termcap	Terminal capability database

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

setenv(1), termcap(5), ttytype(5), stty(1).

NOTES

For compatibility with earlier versions of *tset*, the following flags are accepted and mapped internally as shown:

```
-d type  ->  -m dialup:type
-p type  ->  -m plugboard:type
-a type  ->  -m arpanet:type
```

These flags will disappear eventually.

TSORT(1)

NAME

tsort - topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty - get the terminal's name

SYNOPSIS

tty [-s]

DESCRIPTION

Tty prints the path name of the user's terminal. The **-s** option inhibits printing, allowing one to test just the exit code.

EXIT CODES

0 if standard input is a terminal,
1 otherwise.

DIAGNOSTICS

"not a tty" if the standard input is not a terminal and **-s** is not specified.

NAME

typo - find possible typographical errors

SYNOPSIS

typo [**-n**] [files]

DESCRIPTION

Typo hunts through a document for unusual words, typographic errors, and *hapax legomena* and prints them on the standard output.

The words used in the document are printed out in decreasing order of peculiarity along with an index of peculiarity. An index of 10 or more is considered peculiar. Printing of certain very common English words is suppressed.

The statistics for judging words are taken from the document itself, with some help from known statistics of English. The **-n** option suppresses the help from English and should be used if the document is written in, for example, Urdu.

Troff(1) control lines are ignored. Quote marks, vertical bars, hyphens, and ampersands within words are equivalent to spaces. Words hyphenated across lines are put back together.

FILES

/tmp/ttmp??
/usr/lib/salt
/usr/lib/w2006
/usr/lib/typoprog
/usr/lib/sq2006

SEE ALSO

spell(1).

NAME

umask - set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The octal three digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

NAME

uname - print name of current UNIX

SYNOPSIS

uname [**-snrva**]

DESCRIPTION

Uname prints the current system name of UNIX on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by *uname*(2) to be printed:

-s print the system name (default).

-n print the nodename (the nodename may be a name that the system is known by to a communications network).

-r print the operating system release.

-v print the operating system version.

-a print all the above information.

SEE ALSO

uname(2).

NAME

unget - undo a previous get of an SCCS file

SYNOPSIS

unget [-rSID] **[-s] [-n]** files

DESCRIPTION

Unget undoes the effect of a **get -e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

-r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

-s Suppresses the printout, on the standard output, of the intended delta's *SID*.

-n Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq - report repeated lines in a file

SYNOPSIS

uniq [**-udc** [+n] [-n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing 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*(1). 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 non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units - conversion program

SYNOPSIS

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 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, together with a generous leavening of exotica and a few constants of nature including:

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 U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME

update - periodically update the super block

SYNOPSIS

/etc/update

DESCRIPTION

Update is a daemon that periodically (every 30 seconds) does a *sync* system call. It is executed by **/etc/rc**.

SEE ALSO

sync(2), rc(8).

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [options] ...

DESCRIPTION

Uuclean will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

–d*directory*
Clean *directory* instead of the spool directory.

–p*pre* Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following will cause all files older than the specified time to be deleted.

–n*time* Files whose age is more than *time* hours will be deleted if the prefix test is satisfied. (default time is 72 hours)

–m Send mail to the owner of the file when it is deleted.

This program will typically be started by *cron*(1M).

FILES

/usr/lib/uucp	directory with commands used by *uuclean* internally
/usr/spool/uucp	spool directory

SEE ALSO

uucp(1C), uux(1C).

NAME

uucp, uulog, uuname - unix to unix copy

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

uuname

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 which *uucp* knows about. Shell metacharacters **?*[]** 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 *~/user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;

(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*(2)).

The following options are interpreted by *uucp*:

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-c Use the source file when copying out rather than copying the file to the spool directory (default).

-C Copy the source file to the spool directory.

-m Send mail to the requester when the copy is complete.

-n*user* Notify *user* on the remote system that a file was sent.

-e*sys* Send the *uucp* command to system *sys* to be executed there. (Note - this will only be successful if the remote machine allows the *uucp* command to be executed by **/usr/lib/uucp/uuxqt**.)

Uulog maintains a summary log of *uucp* and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE** by gathering information from partial log files named **/usr/spool/uucp/LOG.*.?**. (These files will only be created if the **LOGFILE** is being used by another process.) It removes the partial log files.

The options cause *uulog* to print logging information:

-s*sys* Print information about work involving system *sys*.

-u*user* Print information about work done for the specified *user*.

Uuname lists the uucp names of known systems. The **-l** option returns the local system name.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uux(1C).
Uucp Implementation Description by D. A. Nowitz.

WARNING

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 **~nuucp** or just **~**).

BUGS

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.)

NAME

uustat - uucp status inquiry and job control

SYNOPSIS

uustat [option] ...

DESCRIPTION

Uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following options are recognized:

-m*mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.

-k*jobn* Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless he is the super-user.

-c*hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.

-u*user* Report the status of all *uucp* requests issued by *user*.

-s*sys* Report the status of all *uucp* requests which communicate with remote system *sys*.

-o*hour* Report the status of all *uucp* requests which are older than *hour* hours.

-y*hour* Report the status of all *uucp* requests which are younger than *hour* hours.

-j*all* Report the status of all the *uucp* requests.

-v Report the *uucp* status verbosely. If this option is not specified, a status code is printed with each *uucp* request.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **-j**, **-m**, **-k**, **-c**, or the rest of other options may be specified.

For example, the command

uustat -uhdc -smhtsa -y72 -v

will print the verbose status of all *uucp* requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
00001	the copy failed, but the reason cannot be determined
00002	permission to access local file is denied
00004	permission to access remote file is denied
00010	bad *uucp* command is generated
00020	remote system cannot create temporary file
00040	cannot copy to remote directory
00100	cannot copy to local directory
00200	local system cannot create temporary file
00400	cannot execute *uucp*
01000	copy succeeded
02000	copy finished, job deleted
04000	job is queued

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/L_stat	system status file
/usr/lib/uucp/R_stat	request status file

SEE ALSO

uucp(1C).
Uustat - A UUCP Status Inquiry Program, by H. Che.

NAME

uusub - monitor uucp network

SYNOPSIS

/usr/lib/uucp/uusub [options]

DESCRIPTION

Uusub defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

-a*sys* Add *sys* to the subnetwork.
-d*sys* Delete *sys* from the subnetwork.
-l Report the statistics on connections.
-r Report the statistics on traffic amount.
-f Flush the connection statistics.
-u*hr* Gather the traffic statistics over the past *hr* hours.
-c*sys* Exercise the connection to the system *sys*. If *sys* is specified as **all**, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

sys #call #ok time #dev #login #nack #other

where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, *#ok* is the number of successful connections, *time* is the the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (e.g. ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (e.g. line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest *uusub* command with the **-u***hr* option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

The command:

uusub -c all -u 24

is typically started by *cron*(1M) once a day.

FILES

/usr/spool/uucp/SYSLOG system log file
/usr/lib/uucp/L_sub connection statistics
/usr/lib/uucp/R_sub traffic statistics

SEE ALSO

uucp(1C), uustat(1C).

NAME

uuto, uupick - public UNIX-to-UNIX file copy

SYNOPSIS

uuto [options] source-files destination
uupick [**-s** system]

DESCRIPTION

Uuto sends *source-files* to *destination*. *Uuto* uses the *uucp*(1C) 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*(1C)). *Logname* 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. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(1) 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* (current directory is default).
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, the public directory, is **/usr/spool/uucppublic**.

SEE ALSO

mail(1), uuclean(1M), uucp(1C), uulog(1C), uuname(1C), uustat(1C), uux(1C).

NAME

uux - unix to unix command execution

SYNOPSIS

uux [-] 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 that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail*(1)) via *uux*.

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.

The - option will cause the standard input to the *uux* command to be the standard input to the *command-string*. For example, the command

uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"

will get the **f1** files from the "usg" and "pwba" machines, execute a *diff* command and put the results in **f1.diff** in the local 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 which are output files, the file name must be escaped using parentheses. For example, the command

uux a!uucp b!/usr/file \(c!/usr/file\)

will send a *uucp* command to system "a" to get **/usr/file** from system "b" and send it to system "c".

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

FILES

/usr/lib/uucp/spool	spool directory
/usr/lib/uucp/*	other data and programs

SEE ALSO

uuclean(1M), uucp(1C).
Uucp Implementation Description by D. A. Nowitz

BUGS

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.

NAME

val - validate SCCS file

SYNOPSIS

val -
val [**-s**] [**-r**SID] [**-m**name] [**-y**type] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a **-**, and named files.

Val has a special argument, **-**, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

-s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

-r*SID* The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e. g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

-m*name* The argument value *name* is compared with the SCCS val.1 keyword in *file*.

-y*type* The argument value *type* is compared with the SCCS keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = can't open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = , **-y** mismatch;
bit 7 = val.1, **-m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned - a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc – version control

SYNOPSIS

vc [**–a**] [**–t**] [**–c**char] [**–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 less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); 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 :.

–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.

DIAGNOSTICS
Use *help*(1) for explanations.

EXIT CODES

0 – normal
1 – any error

NAME

vi - screen oriented (visual) display editor based on ex

SYNOPSIS

/usr/plx/vi [**-t** tag] [**-r**] [+*lineno*] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*(1). *Ex* and *vi* run the same code; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

The *Vi Quick Reference* card and the *Introduction to Display Editing with Vi* provide full details on using *vi.*

FILES

See *ex*(1).

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

ex (1), vi (1), "Vi Quick Reference" card, "An Introduction to Display Editing with Vi".

BUGS

Software tabs using **^T** work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Occasionally inverse video scrolls up into the file from a diagnostic on the last line.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open.*

NAME

volcopy, labelit - copy file systems with label checking

SYNOPSIS

/etc/volcopy [**-bpi**bits-per-inch] [**-feet**size] fsname special1 volname1 special2 volname2

/etc/labelit special [fsname volume [**-n**]]

DESCRIPTION

Volcopy makes a literal copy of the file system using a blocksize matched to the device (10 1024-byte blocks for 800/1600 bpi tape; 88 blocks for everything else). Using *volcopy*, a 2400 foot/1600 bpi 9-track tape will hold a 30000 (1024-byte) block file system, while a cartridge tape will hold a 10000 (1024-byte) block file system. The optional flag arguments are used only with tapes (**-bpi** -- bits-per-inch; **-feet** -- size of reel in feet). The program requests the information if it is not given on the command line. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two drives.

The *fsname* argument represents the mounted name (e.g.: **root**, **u1**, etc.) of the filsystem being copied.

The *special* should be the physical disk section or tape (e.g.: **/dev/rdk1**, **/dev/rmt0**, etc.).

The *volname* is the physical volume name (e.g.: **pk3**, **t0122**, etc.) and should match the external label sticker. Such label names are limited to five or fewer characters.

Special1 and *volname1* are the device and volume from which the copy of the file system is being extracted. *Special2* and *volname2* are the target device and volume.

Fsname and *volname* are recorded in the last 12 characters of the superblock (char fsname[6], volname[6];).

Labelit can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, *labelit* prints current label values. The **-n** option provides for initial labeling of new tapes only (this destroys previous contents).

FILES

/etc/log/filesave a record of file systems/volumes copied

SEE ALSO

fs(5).

BUGS

Only device names beginning **/dev/rmt** are treated as tapes.

Volcopy is extremely slow on the P/25.

NAME

vpmc - compiler for the virtual protocol machine

SYNOPSIS

vpmc [**-m**] [**-r**] [**-c**] [**-x**] [**-s** sfile] [**-l** lfile] [**-i** ifile] [**-o** ofile] file

DESCRIPTION

Vpmc is the compiler for a language that is used to describe communications link protocols. The output of *vpmc* is a load module for the virtual protocol machine (VPM), which is a software construct for implementing communications link protocols (e.g., BISYNC) on the Intelligent Communications Processor (ICP). VPM is implemented by an interpreter in the ICP that cooperates with a driver in the UNIX host computer to transfer data over a communications link in accordance with a specified link protocol. UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal UNIX *open*, *read*, *write*, and *close* operations. The VPM program in the ICP provides error control and flow control using the conventions specified in the protocol.

The language accepted by *vpmc* is essentially a subset of C; the implementation of *vpmc* uses the RATFOR preprocessor (*ratfor*(1)) as a front end; this leads to a few minor differences, mostly syntactic.

Two versions of the interpreter will be available. The appropriate version for a particular application is selected by means of the **-i** option. The BISYNC version (**-i bisync**) supports half-duplex, character-oriented protocols such as the various forms of BISYNC. The HDLC version (**-i hdlc**), which is currently (Sys3 Release 1.1) not supported, supports full-duplex, bit-oriented protocols such as HDLC. The communications primitives used with the BISYNC version are character-oriented and blocking; the primitives used with the HDLC version are frame-oriented and non-blocking.

Options

The meanings of the command-line options are:

-m	Use *m4*(1) instead of *cpp* as the macro preprocessor.
-r	Produce RATFOR output on the standard output and suppress the remaining compiler phases.
-c	Compile only (suppress the assembly and linking phases).
-x	Retain the intermediate files used for communication between passes.
-s *sfile*	Save the generated VPM assembly language on file *sfile*.
-l *lfile*	Produce a VPM assembly-language listing on file *lfile*.
-i *ifile*	Use the interpreter version specified by *ifile* (default **bisync**).
-o *ofile*	Write the executable object file on file *ofile* (default **a.out**).

These options may be given in any order.

Programs

Input to *vpmc* consists of a (possibly null) sequence of array declarations, followed by one or more function definitions. The first defined function is invoked (on command from the UNIX VPM driver) to begin program execution.

Functions

A function definition has the following form:

function *name*( )
statement_list
end

Function arguments (formal parameters) are not allowed. The effect of a function call with arguments can be obtained by invoking the function via a macro that first assigns the value of each argument to a global variable reserved for that purpose. See *EXAMPLES* below.

A *statement_list* is a (possibly null) sequence of labeled statements. A *labeled_statement* is a statement preceded by a (possibly null) sequence of labels. A *label* is either a name followed by a colon (:) or a decimal integer optionally followed by a colon.

The statements that make up a statement list must be separated by semicolons (;). (A semicolon at the end of a line can usually be omitted; refer to the description of RATFOR for details.) Null statements are allowed.

Statement Syntax

The following types of statements are allowed:

expression
lvalue=*expression*
lvalue+=*expression*
lvalue-=*expression*
lvalue |=*expression*
lvalue&=*expression*
lvalue^=*expression*
lvalue <<=*expression*
lvalue >>=*expression*
if(*expression*)*statement*
if(*expression*)*statement* else *statement*
while(*expression*)*statement*
for(*statement*; *expression*; *statement*)*statement*
repeat *statement*
repeat *statement* until *expression*
break
next
switch(*expression*){*case_list*}
return(*expression*)
return
goto *name*
goto *decimal_constant*
{*statement_list*}

repeat is equivalent to the **do** keyword in C; **next** is equivalent to **continue**.

A *case_list* is a sequence of statement lists, each of which is preceded by a label of the form:

case *constant*:

The label for the last *statement_list* in a *case_list* may be of the form:

default:

Unlike C, RATFOR supplies an automatic **break** preceding each new case label.

Expression Syntax

A *primary_expression* (abbreviated *primary*) is an lvalue or a constant. An *lvalue* is one of the following:

name
name[*constant*]

A *unary_expression* (abbreviated *unary*) is one of the following:

primary
name()
system_call
++*lvalue*
--*lvalue*

(expression)
!unary
~unary

The following types of expressions are allowed:

unary
unary+primary
unary-primary
unary|primary
unary&primary
unary&~primary
unary^primary
unary<<primary
unary>>primary
unary==primary
unary!=primary
unary>primary
unary<primary
unary>=primary
unary<=primary

Note that the right operand of a binary operator can only be a constant, a name, or a name with a constant subscript.

System Calls

A VPM program in the ICP (a compiled protocol script) interacts with a communications device and a driver in the host computer by means of system calls (primitives). These system calls are part of the ICP operating system library.

The following primitives are available only in the BISYNC version of the interpreter:

crc16(*primary***)**

The value of the primary expression is combined with the cyclic redundancy check-sum at the location passed by a previous **crcloc** system call. The CRC-16 polynomial

$$(x^{16}+x^{15}+x^2+1)$$

is used for the check-sum calculation.

crcloc(*name***)**

The two-byte array starting at the location specified by *name* is cleared. The address of the array is recorded as the location to be updated by subsequent **crc16** system calls.

get(*lvalue***)**

Get a byte from the current *transmit* buffer. The next available byte, if any, is copied into the location specified by *lvalue*. The returned value is zero if a byte was obtained, otherwise it is non-zero.

getrbuf(*name***)**

Get (open) a *receive* buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a receive buffer has previously been opened via a **getrbuf** call but has not yet been closed via a call to **rtnrbuf**, that buffer is reinitialized and remains the current buffer.

getxbuf(*name***)**

Get (open) a *transmit* buffer. The returned value is zero if a buffer is available,

otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a transmit buffer has previously been opened via a **getxbuf** call but has not yet been closed via a call to **rtnxbuf**, that buffer is reinitialized and remains the current buffer.

put(*primary***)**
Put a byte into the current *receive* buffer. The value of the primary expression is inserted into the next available position, if any, in the current receive buffer. The returned value is zero if a byte was transferred, otherwise it is non-zero.

rcv(*lvalue***)**
Receive a character. The process delays until a character is available in the input silo. The character is then moved to the location specified by *lvalue* and the process is reactivated.

rsom(*constant***)**
Skip to the beginning of a new *receive* frame. The receiver hardware is cleared and the value of *constant* is stored as the receive sync character. This call is used to synchronize the local receiver and remote transmitter when the process is ready to accept a new receive frame.

rtnrbuf(*name***)**
Return a *receive* buffer. The original values of the buffer parameters for the current receive buffer are replaced with values from the array specified by *name*. The current receive buffer is then released to the driver.

rtnxbuf(*name***)**
Return a *transmit* buffer. The original values of the buffer parameters for the current transmit buffer are replaced with values from the array specified by *name*. The current transmit buffer is then released to the driver.

xeom(*constant***)**
Transmit end-of-message. The value of the constant is transmitted, then the transmitter is shut down.

xmt(*primary***)**
Transmit a character. The value of the primary expression is transmitted over the communications line. If the output silo is full, the process waits until there is room in the silo.

xsom(*constant***)**
Transmit start-of-message. The transmitter is cleared, then the value of *constant* is transmitted six times. This call is used to synchronize the local transmitter and the remote receiver at the beginning of a frame.

The following primitives are available with all versions of the interpreter:

dsrwait( )
Wait for modem-ready and then set modem-ready mode. The process delays until the modem-ready signal from the modem interface is asserted. If the modem-ready signal subsequently drops, the process is terminated. If **dsrwait** is never invoked, the modem-ready signal is ignored.

exit(*primary***)**
Terminate execution. The process is halted and the value of the primary expression is passed to the driver.

getcmd(*name***)**
Get a command from the driver. If a command has been received from the driver since the last call to **getcmd**, four bytes of command information are copied into the array

specified by *name* and a value of **true** (non-zero) is returned. If no command is available, the returned value is **false** (zero).

pause( )
Return control to the dispatcher. This primitive informs the dispatcher that the virtual process may be suspended until the next occurrence of an event that might affect the state of the protocol for this line. Examples of such events are: (1) completion of an output transfer, (2) completion of an input transfer, (3) timer expiration, and (4) a buffer-in command from the driver. In a multi-line implementation, the **pause** primitive allows the process for a given line to give up control to allow the processor to service another line.

rtnrpt(*name***)**
Return a report to the driver. Four bytes from the array specified by *name* are transferred to the driver. The process delays until the transfer is complete.

testop(*primary***)**
Test for odd parity. The returned value is **true** (non-zero) if the value of the primary expression has odd parity, otherwise the returned value is **false** (zero).

timeout(*primary***)**
Schedule or cancel a timer interrupt. If the value of the *primary* expression is non-zero, the current values of the program counter and stack pointer are saved and a timer is loaded with the value of *primary*. The system call then returns immediately with a value of **false** (zero) as the returned value. The timer is decremented each tenth of a second thereafter. If the timer is decremented to zero, the saved values of the program counter and stack pointer are restored and the system call returns with a value of **true** (non-zero). The effect of the timer interrupt is to return control to the code immediately following the **timeout** system call, at which point a non-zero return value indicates that the timer has expired. The **timeout** system call with a non-zero argument is normally written as the condition part of an **if** statement. A **timeout** system call with a zero argument value cancels all previous **timeout** requests, as does a **return** from the function in which the **timeout** system call was made. A **timeout** system call with a non-zero argument value overrides all previous **timeout** requests. The maximum permissible value for the argument is 255, which gives a timeout period of 25.5 seconds.

timer(*primary***)**
Start a timer or test for timer expiration. If the value of the *primary* is non-zero, a software timer is loaded with the value of the *primary* and a value of **true** (non-zero) is returned. The timer is decremented each tenth of a second thereafter until it reaches zero. If the value of the *primary* is zero, the returned value is the current value of the timer; this will be **true** (non-zero) if the value of the timer is currently non-zero, otherwise **false** (zero). The timer used by this primitive is different from the timer used by the **timeout** primitive.

trace(*primary***[,***primary***])**
The values of the two primary expressions and the current value of the script location counter are passed to the driver. If the second *primary* is omitted, a zero is used instead. The process delays until the values have been accepted by the host computer.

Constants

A *constant* is a decimal, octal, or hexadecimal integer, or a single character enclosed in single quotes. A token consisting of a string of digits is taken to be an octal integer if the first digit is a zero, otherwise the string is interpreted as a decimal integer. If a token begins with **0x** or **0X**, the remainder of the token is interpreted as a hexadecimal integer. The hexadecimal digits include **a** through **f** or, equivalently, **A** through **F**.

Variables

Variable names may be used without having been previously declared. All names are global. All values are treated as 8-bit unsigned integers.

Arrays of contiguous storage may be allocated using the **array** declaration:

array *name*[*constant*]

where *constant* is a decimal integer. Elements of arrays can be referenced using constant subscripts:

name[*constant*]

Indexing of arrays assumes that the first element has an index of zero.

Names

A *name* is a sequence of letters and digits; the first character must be a letter. Upper- and lower-case letters are considered to be distinct. Names longer than 31 characters are truncated to 31 characters. The underscore (_) may be used within a name to improve readability, but is discarded by RATFOR.

Preprocessor Commands

If the **-m** option is omitted, comments, macro definitions, and file inclusion statements are written as in C. Otherwise, the following rules apply:

1. If the character # appears in an input line, the remainder of the line is treated as a comment.
2. A statement of the form:

   define(*name*,*text*)

   causes every subsequent appearance of *name* to be replaced by *text*. The defining text includes everything after the comma up to the balancing right parenthesis; multi-line definitions are allowed. Macros may have arguments. Any occurrence of **$***n* within the replacement text for a macro will be replaced by the *n*th actual argument when the macro is invoked.
3. A statement of the form:

   include(*file*)

   inserts the contents of *file* in place of the **include** command. The contents of the included file is often a set of definitions.

EXAMPLES

These examples require the use of the **-m** option.

```
# The function defined below transmits a frame in transparent BISYNC.
# A transmit buffer must be obtained with getxbuf before the function
# is invoked.
#
# Define symbolic constants:
#
define(DLE,0x10)
define(ETB,0x26)
define(PAD,0xff)
define(STX,0x02)
define(SYNC,0x32)
#
# Define a macro with an argument:
#
define(xmtcrc,{crc16($1); xmt($1);})
```

```
#
# Declare an array:
#
array crc[2];
#
# Define the function:
#
function xmtblk( )
        crcloc(crc);
        xsom(SYNC);
        xmt(DLE);
        xmt(STX);
        while(get(byte)==0){
                if(byte == DLE)
                        xmt(DLE);
                xmtcrc(byte);
        }
        xmt(DLE);
        xmtcrc(ETB);
        xmt(crc[0]);
        xmt(crc[1]);
        xeom(PAD);
end
#
# The following example illustrates the use of macros to simulate a
# function call with arguments.
#
# The macro definition:
#
define(xmtctl,{c=$1;d=$2;xmtctl1( )})
#
# The function definition:
#
function xmtctl1( )
        xsom(SYNC);
        xmt(c);
        if(d!=0)
                xmt(d);
        xeom(PAD);
end
#
# Sample invocation:
#
function test( )
        xmtctl(DLE,0x70);
end
```

FILES

sas_temp*	temporaries
/tmp/sas_ta??	temporary
/tmp/sas_tb??	temporary
/usr/lib/vpm/pass*	compiler phases
/usr/lib/vpm/pl	compiler phase

/usr/lib/vpm/vratfor	compiler phase
/lib/cpp	preprocessor
/usr/bin/m4	preprocessor

SEE ALSO

m4(1), ratfor(1), vpmstart(1C), vpm(4).
C Reference Manual by D. M. Ritchie.
RATFOR-A Preprocessor for a Rational Fortran by B. W. Kernighan.
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.
Software Tools by B. W. Kernighan and P. J. Plauger (pp. 28-30).

NAME

vpmstart, vpmsnap, vpmtrace - load the ICP print VPM traces

SYNOPSIS

vpmstart device n [filen]

vpmsnap

vpmtrace

DESCRIPTION

Vpmstart writes *filen* (**a.out** by default) to the Intelligent Communications Processor (ICP) specified by *device*.

The argument *n* is a magic number that the ICP driver saves to identify the running program. This number is checked when the VPM driver is opened to provide some assurance that the program running in the ICP is the one expected. The magic number for VPM interpreters is 6. When *filen* has been written to the ICP its execution is begun. *Filen* may be any file executable by the ICP.

If *filen* is made using *vpmc*(1C), the VPM interpreter will be started by *vpmstart*. The VPM interpreter waits for a RUN command from the VPM driver before beginning execution of the protocol script. The RUN command is sent by the VPM driver when the corresponding VPM device file is opened.

Vpmsnap opens the trace driver (minor device number 1) and reads and prints time-stamped event records until killed.

Vpmtrace opens the trace driver (minor device number 0) and reads and prints event records until killed.

SEE ALSO

vpmc(1C), trace(4), vpm(4).

NAME

vtty - connect to a remote host via NOS

SYNOPSIS

vtty host

DESCRIPTION

Vtty establishes a *cu*(1) link between one UNIX system and another. Both systems must be running the Plexus Network Operating System (NOS).

Vtty scans the file **/usr/lib/nos/vtconf** for the *host* name specified, and remembers the minor device number(s) associated with that host. It then searches the directory **/dev** for special files that have that minor device number. If it finds entries in **/dev** with the appropriate minor device number, it checks **/usr/spool/uucp** for lock files associated with these devices, and initiates a *cu* process with the first available device.

All normal setup for *cu* and the virtual terminal facility of NOS must be done in order for this command to work. In particular, the file **/usr/lib/uucp/L-devices** must be set up properly, and virtual terminal ports must be established in **/dev** and configured in **/usr/lib/nos/vtconf**.

FILES

/usr/lib/nos/vtconf	virtual terminal configuration file
/usr/spool/uucp	*cu* lock files here
/dev/*	virtual terminal ports located here

EXAMPLE

Assuming the system **plx** is running NOS, and one or more virtual terminal ports have been established for **plx** in **/dev**, and **/usr/lib/nos/vtconf** knows about **plx**, you can connect to **plx** by typing

$ vtty plx

If successful, the message

Connected

appears, followed by the normal login prompt.

DIAGNOSTICS

Cannot open /dev
The directory **dev** cannot be accessed.

Cannot open /usr/lib/nos/vtconf
The file **/usr/lib/nos/vtconf** cannot be opened or does not exist.

<name> not found in vtconf
The host name *name* is not a configured host.

No vt device available. Try again.
All ports are in use. Try again later.

BUGS

There is no semaphore to prevent collisions between *vtty* requests. Hence the error message indicating lack of devices may be erroneous.

NAME

wait - await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus can't be waited for.

NAME

wall - write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message to all currently logged in users preceded by "Broadcast Message from ...". It is used to warn all users, typically prior to shutting down the system.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

"Cannot send to ..." when the open on a user's tty file fails.

NAME

wc - word count

SYNOPSIS

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 new-lines.

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 will be printed along with the counts.

NAME

what - identify SCCS files

SYNOPSIS

what files

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for @(#) (this is **@(#)** at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the SCCS command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

It's possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

who - who is on the system

SYNOPSIS

who [who-file] [**am I**]

DESCRIPTION

Who, without an argument, lists the login name, terminal name, and login time for each current UNIX user.

Without an argument, *who* examines the **/etc/utmp** file to obtain its information. If a file is given, that file is examined. Typically the given file will be **/usr/adm/wtmp**, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with **/dev/** suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with **x** in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in **who am I** (and also **who are you**), *who* tells who you are logged in as.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5).

NAME

whodo - who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

Whodo produces merged, reformatted, and dated output from the *who*(1) and *ps*(1) commands.

SEE ALSO

ps(1), who(1).

NAME

write - write to another user

SYNOPSIS

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* writes **EOF** 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*(1) command. At the outset, writing is allowed. Certain commands, in particular *nroff*(1) and *pr*(1), 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(1), mesg(1), who(1).

NAME

xargs - construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed *initial-arguments* with arguments read from 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 one's **$PATH**. 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 new-lines; 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 then *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 non-empty *number* lines of arguments from standard input. 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 new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted 1 is assumed. Option **-x** is forced.

-i*replstr* — Insert mode: *command* is executed for each line from 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* — Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be 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* arguments must fit in the *size* limitation, else *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 asked whether to execute *command* each invocation. Trace mode (**-t**) is turned on to print 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 just 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. Underbar (_) is assumed for the logical **EOF** string if **-e** is not coded. **-e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

Xargs will terminate if either it 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*(1)) 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 asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1. ls | xargs -p -l ar r arch
2. ls | xargs -p -l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

DIAGNOSTICS

Self explanatory.

NAME

xref - cross reference for C programs

SYNOPSIS

xref [file ...]

DESCRIPTION

Xref reads the named *files* or the standard input if no file is specified and prints a cross reference consisting of lines of the form

identifier file-name line-numbers ...

Function definition is indicated by a plus sign (+) preceding the line number.

SEE ALSO

cref(1).

NAME

xstr - extract strings from C programs to implement shared strings

SYNOPSIS

/usr/plx/xstr [**-c**] [**-**] [file]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

xstr -c name

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An approporiate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

xstr

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

xstr name

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '-' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

cc -E name.c | **xstr -c** -
cc -c x.c
mv x.o name.o

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' doesn't touch *strings*

NOTES

This command is based on a similar one from the University of California at Berkeley.

SEE ALSO

mkstr(1)

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc - yet another compiler-compiler

SYNOPSIS

yacc [**-vd**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*. A typical load line is the following

cc y.tab.c -ly

If the **-v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **-d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs
/usr/lib/liby.a	yacc library

SEE ALSO

lex(1).
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC - Yet Another Compiler Compiler by S. C. Johnson.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.