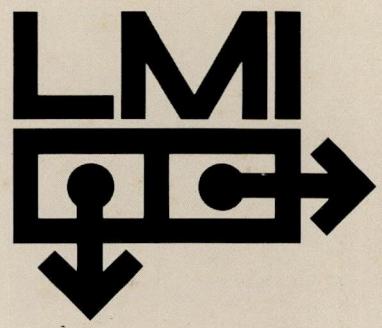

LAMBDA

Programmer's Manual, Volume 1. Section 2

# Nu Machine Release and Update Information

**Distribution 3**

**TI-2242801-0001**

**November, 1983**

Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA

SECTION 1
INTRODUCTION

This document contains a summary of Distribution 3 contents, procedures to update systems running Distribution 2 software to Distribution 3, and procedures to do a complete re-installation of Distribution 3 on a disk which has become totally corrupted.

**New Rack and Office Systems are shipped with Distribution 3 already installed; no software installation is required.**

The following manuals should be read before proceeding with and update or installation procedures:

**Nu Machine Installation and User Manual**

**Nu Machine Operating System User Guide**

In addition, the following manuals can be an aid to understanding the procedures and commands used in this manual.

**Nu Machine SDU Monitor User's Manual**

**Nu Machine UNIX^ Programmer's Manual**

**WARNING: UPDATING TO DISTRIBUTION 3 SOFTWARE INVOLVES COMPLETE DESTRUCTION OF DATA ON SOME DISK PARTITIONS. ANY USER FILES SHOULD BE SAVED ON TAPE BEFORE STARTING THE SOFTWARE INSTALLATION PROCEDURE.**

^UNIX is a trademark of Bell Telephone Laboratories, Incorporated.

SECTION 1
DISTRIBUTION 3 SUMMARY

The following is a summary of the changes included in the Distribution 3 package:

1. Kernel, driver and command bug fixes to increase stability of the system.

2. New commands: ac, cal, comm, f77, iostat, lpr, spell, split, tbl, and tip.

3. An optional change to increase the disk interleave factor. This change will increase performance on a loaded system.

4. Updates to the Nu Machine diagnostics.

5. The disk partitioning has been modified to allow a separate small file system for the SDU which contains SDU commands, diagnostics, and drivers. The new partition which has been added is **/dev/dkØg**. The UNIX root is still **/dev/dkØa**. This was accomplished by splitting the old UNIX root into 2 pieces. The starting positions for other disk partitions has **NOT** changed.

6. As a result of the addition of **/dev/dkØg**, the /monitor directory is now in the SDU file system (instead of the UNIX root file system) and is referenced by **/sdu/monitor**.

7. Under the SDU Monitor, **/disk** will reference **/dev/dkØg** (not the UNIX root). The UNIX root is accessed by using a new SDU driver called **uroot**. The default path name used by **uboot** has been changed to **/uroot/unix** from the old **/disk/unix**.

8. New manuals:

   **Nu Machine UNIX Programmer's Volume 2B**

   **Nu Machine Operating System User Guide**

9. No changes to the SDU Monitor.

## SECTION 2
## DISTRIBUTION TAPE CONTENTS

The Distribution 3 software is provided on 2 1/2" tapes for Rack Systems, or 2 1/4" tapes for Office Systems. The 1/2" and 1/4" versions of each tape contain the same files, only the media is different. A set of 1/2" tapes can be used with and Office system if it has a 1/2" tape drive and vice-versa for 1/4" tapes.

The first tape is the **Nu Machine Diagnostic** tape. It contains 3 files:

**file 1:**
Tar driver for the SDU Monitor.

**file 2:**
Tar of diagnostics.

**file 3:**
A file system image of the SDU file system which contains diagnostics, SDU commands, and SDU drivers.

The second tape is the **Nu Machine UNIX Distribution** tape. It contains 2 files:

**file 1:**
UNIX root file system image.

**file 2:**
Tar of /usr.

SECTION 3
UPDATING TO DISTRIBUTION 3

This section contains procedures for updating systems which are currently running Distribution 2 software. If the system is not now running Distribution 2 software or if the disk has been corrupted, then see Section 4 for full installation procedures. **This procedure is ONLY for update purposes.**

If the new disk interleave factor is desired, then the disk must be reformatted. See Section 4 and follow the full installation procedures. The change of disk interleave factor is **OPTIONAL.** If this new interleave is **NOT** desired, then continue the update procedures and **DO NOT RE-FORMAT THE DISK.**

The update procedure involves replacement of the UNIX root file system (/dev/dkØa), addition of the SDU file system (/dev/dkØg), and update of /usr. Be sure to save any user added or user modified files which are in the root. Examples are /etc/passwd, /etc/motd, and /etc/ident. Although this procedure is non-destructive to user files on /usr, as a safety precaution, it is recommended that ALL user added or modified files in all file systems be backed up before preceding.

**New Rack and Office Systems are shipped with Distribution 3 already installed; no software installation is required.**

IMPORTANT: UPDATING TO DISTRIBUTION 3 INVOLVES DATA DESTRUCTION ON SOME DISK PARTITIONS. BACKUP YOUR LOCAL FILES VIA TAR BEFORE STARTING ANY OF THE INSTALLATION PROCEDURES.

If the system is to be upgraded to Distribution 3, the entire procedure must be followed. Do not mix any part of the new software with old software.

**SECTION 3.1**
**RACK SYSTEM UPDATE**

This update procedure applies to systems with Fujitsu Eagle disk drives using 1/2" tape drive.

1. Bring down the currently running UNIX to the SDU Monitor (be sure to sync first). SDU Monitor should respond with a >> prompt.

2. Load the Nu Machine Diagnostic tape.

3. Setup the CMOS ram. The contents of CMOS RAM have changed to reflect the new disk partition. This step **MUST** be performed even if the **SETUP** light is not on.

```
>> init
>> /tar/setup clear
>> /tar/setup eagle sp shell
```

4. Reset the system and check that the **SETUP** light goes out and the **RUN** light is on. Do not continue with the installation procedure if the **RUN** light is not on.

5. Copy the SDU file system image from tape to disk via:

```
>> init
>> copy tape/3 $disk
```

   The SDU Monitor should respond with "1047552 bytes copied".

6. Load the Nu Machine UNIX Distribution tape.

7. Copy the UNIX root file system image from tape to disk via:

```
>> init
>> cp tape/1 $disk 1023 -6942
```

   The SDU Monitor should respond with "cp finished: 7108608 bytes copied". In some cases the message "process got multibus timeout" may appear. This message can be ignored in this case only.

8. Boot UNIX via (assumes switch setting 3):

```
>> init
>> uboot -k rsd -C /disk/monitor/conf/std.eagle
```

   If switch setting 1 is being used, change **rsd** to **ttya** in the uboot command. As an alternative, or for non-

standard board configurations the auto-configuration option can be used with uboot:

```
>> init
>> uboot -a -k rsd
```

9. Single-user UNIX should now be running. Several links must be added as follows:

```
# /etc/mount /dev/dkØg /sdu
# cd /sdu/monitor/conf
# ln std.eagle std
# cd /etc
# ln rc.eagle rc
# ln fstab.eagle fstab
# ln checklist.eagle checklist
# sync
# /etc/umount /dev/dkØg
```

1Ø. Set the date, sync the disk, and go multi-user via:

```
# date yymmddhhmm.ss
# sync
# ^D
```

11. Login as root and un-tar /usr via:

```
-> cd /usr
-> tar xpvf /dev/rmtØ
```

**NOTE:** If for any reason the tape has been moved since the UNIX root was copied from it, or if the tar failed an you want to retry it, you will need to reposition the tape to file 2 via:

```
-> dd if=/dev/rmtØ of=/dev/null count=1
-> dd if=/dev/rmt4 of=/dev/null bs=1k
```

**SECTION 3.2**
**OFFICE SYSTEM UPDATE**

This update procedure applies to systems with Fujitsu Micro disk drives using 1/4" tape drive.

1. Bring down the currently running UNIX to the SDU Monitor (be sure to sync first). SDU Monitor should respond with a >> prompt.

2. Load the Nu Machine Diagnostic tape.

3. Setup the CMOS ram. The contents of CMOS RAM have changed to reflect the new disk partition. This step **MUST** be performed even if the **SETUP** light is not on.

```
>> init
>> /tarq/setup clear
>> /tarq/setup micro sp shell
```

4. Reset the system and check that the **SETUP** light goes out and the **RUN** light is on. Do not continue with the installation procedure if the **RUN** light is not on.

5. Copy the SDU file system image from tape to disk via:

```
>> init
>> copy quart/3 $disk
```

   The SDU Monitor should respond with "1047552 bytes copied".

6. Load the Nu Machine UNIX Distribution tape.

7. Copy the UNIX root file system image from tape to disk via:

```
>> init
>> cp quart/1 $disk 1023 -6942
```

   The SDU Monitor should respond with "cp finished: 7108608 bytes copied". In some cases the message "process got multibus timeout" may appear. This message can be ignored in this case only.

8. Boot UNIX via (assumes switch setting 3):

```
>> init
>> uboot -k rsd -C /disk/monitor/conf/std.micro
```

   If switch setting 1 is being used, change **rsd** to **ttya** in the uboot command. As an alternative, or for non-

standard board configurations the auto-configuration option can be used on uboot:

```
>> init
>> uboot -a -k rsd
```

9. Single-user UNIX should now be running. Several links must be added as follows:

```
# /etc/mount /dev/dkØg /sdu
# cd /sdu/monitor/conf
# ln std.micro std
# cd /etc
# ln rc.micro rc
# ln fstab.micro fstab
# ln checklist.micro checklist
# sync
# /etc/umount /dev/dkØg
```

10. Set the date, sync the disk, and go multi-user via:

```
# date yymmddhhmm.ss
# sync
# ^D
```

11. Login as root and un-tar /usr via:

```
-> cd /usr
-> qtrop bot
-> qtrop rdmark
-> tar xpvf /dev/rqtØ
```

## SECTION 4
## COMPLETE INSTALLATION

This section contains full installation procedures for Distribution 3. These procedures are used to restore a system disk when it has become totally corrupted, or to change disk interleave factor. For update of systems currently running Distribution 2, see Section 3.

**New Rack and Office Systems are shipped with Distribution 3 already installed; no software installation is required.**

**IMPORTANT: INSTALLING DISTRIBUTION 3 INVOLVES DATA DESTRUCTION ON SOME DISK PARTITIONS AND MAY INVOLVE REFORMATTING THE DISK. BACKUP YOUR LOCAL FILES VIA TAR BEFORE STARTING ANY OF THE INSTALLATION PROCEDURES.**

If the Distribution 3 software is to be installed, the entire procedure must be followed. Do not mix any part of the new software with old software.

SECTION 4.1
RACK SYSTEM INSTALLATION

The following procedure is for installation of Distribution 3 software on Rack systems with Fujitsu Eagle disk drive.

1. Power up the system, select switch setting 3, and reset. The SDU Monitor should respond with a >> prompt on the **rsd**.

   **NOTE:** If the **SETUP** light remains on after the reset, then the SDU defaults to Mode Switch position Ø. In this case only, a terminal must be connected to the SDU remote serial port (ttya) and be set to **3ØØ baud** to do step 4.

   If an **rsd** is not available, a terminal connected to **ttya** can be used by selecting switch setting 1. However, when **SETUP** remains on after reset, the terminal should be set to 3ØØ baud. After setup is complete and the system is reset, the terminal should be set to 96ØØ baud.

2. Load the Nu Machine Diagnostics tape.

3. Setup the CMOS ram via:

```
>> init
>> /tar/setup clear
>> /tar/setup eagle sp shell
```

4. Reset the system and check that the **SETUP** light goes out and the **RUN** light is on.

   Do not continue with the installation procedure if the **RUN** light is not on. The prompt should now appear on the rsd if switch setting 3 is used, or on ttya if switch setting 1 is used. The terminal on ttya must now be set to 96ØØ baud.

5. If the state of the disk is unknown or the new interleave factor is desired, then the disk must be reformatted. If the disk was already formatted and was running Distribution 2 software, then this step may be skipped. Reformat the disk via:

```
>> /tar/2181 -tvDF
```

6. Copy the SDU file system image from tape to disk via:

```
>> init
>> copy tape/3 $disk
```

The SDU Monitor should respond with "1047552 bytes copied".

7. Load the Nu machine UNIX Distribution tape.

8. Copy the UNIX root file system image to disk via:

```
>> init
>> cp tape/1 $disk 1023 -6942
```

The SDU Monitor should respond with "cp finished: 7108608 bytes copied". In some cases the message "process got multibus timeout" may appear. This message can be ignored in this case only.

9. Boot UNIX via (assumes switch setting 3):

```
>> init
>> uboot -k rsd -C /disk/monitor/conf/std.eagle
```

If switch setting 1 is being used, change **rsd** to **ttya** in the uboot command. As an alternative, or for non-standard board configurations the auto-configuration option can be used on uboot:

```
>> init
>> uboot -a -k rsd
```

10. Single-user UNIX should now be running. Build new file systems via:

```
# csh makedisk eagle
```

**CAUTION:** makedisk contains commands to make file systems (mkfs). This results in total loss of any files that were on the disk. Makedisk should only be run once during the installation procedure. When installation is complete, makedisk should be removed to ensure that is is not accidentally run.

11. Set the date, sync the disk, and go multi-user via:

```
# date yymmddhhmm.ss
# sync
# ^D
```

12. Login as root and un-tar /usr via:

```
-> cd /usr
-> tar xpvf /dev/rmt0
```

**NOTE:** If for any reason the tape has been moved since

the root was copied from it, or if the tar failed an you want to retry it, you will need to reposition the tape to file 2 via:

```
-> dd if=/dev/rmtØ of=/dev/null count=1
-> dd if=/dev/rmt4 of=/dev/null bs=1k
```

SECTION 4.2
OFFICE SYSTEM INSTALLATION

The following procedure is for installation of Distribution 3 software on Office systems with Fujitsu Micro disk drive.

1. Power up the system, select switch setting 3, and reset. The SDU Monitor should respond with a >> prompt on the **rsd**.

   **NOTE:** If the **SETUP** light remains on after the reset, then the SDU defaults to Mode Switch position Ø. In this case only, a terminal must be connected to the SDU remote serial port (ttya) and be set to **3ØØ baud** to do step 4.

   If an **rsd** is not available, a terminal connected to **ttya** can be used by selecting switch setting 1. However, when **SETUP** remains on after reset, the terminal should be set to 3ØØ baud. After setup is complete and the system is reset, the terminal should be set to 96ØØ baud.

2. Load the Nu Machine Diagnostics tape.

3. Setup the CMOS ram via:

```
>> init
>> /tarq/setup clear
>> /tarq/setup micro sp shell
```

4. Reset the system and check that the **SETUP** light goes out and the **RUN** light is on.

   Do not continue with the installation procedure if the **RUN** light is not on. The prompt should now appear on the rsd if switch setting 3 is used, or on ttya if switch setting 1 is used. The terminal on ttya must now be set to 96ØØ baud.

5. If the state of the disk is unknown or the new interleave factor is desired, then the disk must be reformatted. If the disk was already formatted and was running Distribution 2 software, then this step may be skipped. Reformat the disk via:

```
>> /tarq/2181 -tvDF
```

6. Copy the SDU file system image from tape to disk via:

```
>> init
>> copy quart/3 $disk
```

The SDU Monitor should respond with "1047552 bytes copied".

7. Load the Nu machine UNIX Distribution tape.

8. Copy the UNIX root file system image to disk via:

```
>> init
>> cp quart/1 $disk 1023 -6942
```

The SDU Monitor should respond with "cp finished: 7108608 bytes copied". In some cases the message "process got multibus timeout" may appear. This message can be ignored in this case only.

9. Boot UNIX via (assumes switch setting 3):

```
>> init
>> uboot -k rsd -C /disk/monitor/conf/std.micro
```

If switch setting 1 is being used, change **rsd** to **ttya** in the uboot command. As an alternative, or for non-standard board configurations the auto-configuration option can be used on uboot:

```
>> init
>> uboot -a -k rsd
```

10. Single-user UNIX should now be running. Build new file systems via:

```
# csh makedisk micro
```

**CAUTION:** makedisk contains commands to make file systems (mkfs). This results in total loss of any files that were on the disk. Makedisk should only be run once during the installation procedure. When installation is complete, makedisk should be removed to ensure that is is not accidentally run.

11. Set the date, sync the disk, and go multi-user via:

```
# date yymmddhhmm.ss
# sync
# ^D
```

12. Login as root and un-tar /usr via:

```
-> cd /usr
-> qtrop bot
-> qtrop rdmark
-> tar xpvf /dev/rqt0
```

## SECTION 5
## TROUBLESHOOTING

If there are problems installing the software system, an example of a troubleshooting flow is contained in the Nu Machine Operating System User Guide. If further problems exist, contact Customer Technical Services, Texas Instruments, Irvine, (714) 660-8217.

# Nu Machine UNIX Programmer's Manual

**Volume 1, Section 1**

**TI-2242803-0001**

**November, 1983**

Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA

## NAME

intro - introduction to commands

## DESCRIPTION

This section describes publicly accessible commands in alphabetic order. 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.

## DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of `normal' termination) one supplied by the program, see wait and exit(2). The former byte is Ø for normal termination, the latter is customarily Ø for successful execution, nonzero 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.

## NAME

a86 - assembler for the Intel iAPX86

## SYNOPSIS

a86 [ option ] file

## DESCRIPTION

A86 assembles the named file.

The following options are interpreted by a86.

-o output
: Name the output file output. If this option is not specified, the output file is named file.b86, where file is the name of the input file.

-l
: Create a listing file named file.list, where file is the name of the input file.

-s
: Create a listing of the symbol table on the list file.

-b
: Perform span dependent instruction optimization.

## SEE ALSO

ld86(1), nm86(1), b.out(5)

## DIAGNOSTICS

When syntactic or semantic errors occur, the line that the error occurred on and an error number is printed. Following all of the lines in error, a summary of the error numbers and the text of the error message is printed. If a listing file (-l) option was specified, lines in error are not echoed -- examine the list file to determine the problems.

**NAME**

ac - login accounting

**SYNOPSIS**

ac [ -w wtmp ] [ -p ] [ -d ] [ people ] ...

**DESCRIPTION**

Ac produces a printout giving connect time for each user who has logged in during the life of the current wtmp file. A total is also produced. -w is used to specify an alternate wtmp file. -p prints individual totals; without this option, only totals are printed. -d causes a printout for each midnight to midnight period. Any people will limit the printout to only the specified login names. If no wtmp file is given, /usr/adm/wtmp is used.

The accounting file /usr/adm/wtmp is maintained by init and login. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length Ø. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

**FILES**

/usr/adm/wtmp

**SEE ALSO**

init(8), login(1), 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 Ø. 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 Ø; 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 rØ ... r5 sp pc ps.

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 disinguish 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.

(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.

/f Locations starting at address in corfil are printed according to the format f.

=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 temporarily 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 Ø. |
| 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.
C 1 Print the addressed character using the following escape convention. Character values ØØØ to Ø4Ø are printed as @ followed by the corresponding character in the range Ø1ØØ to Ø14Ø. 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(3)).
i n Print as 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.
a Ø 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 Ø When preceded by an integer tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop.
r Ø Print a space.
n Ø Print a newline.
"..." Ø
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.

newline
If the previous command temporarily incremented dot, make the increment permanent. 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 floating registers in single or double length. If the floating point status of ps is set to double (0200 bit) then double length is used anyway.
- b Print all breakpoints and their associated counts and commands.
- a ALGOL 68 stack backtrace. If address is given then it is taken to be the address of the current frame (instead of r4). If count is given then only the first count frames are printed.
- c C stack backtrace. If address is given then it is taken as the address of the current frame (instead of r5). If C is used then the names and (16 bit) values of all automatic and static variables are printed for each active function. 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.
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.

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 c 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<address<e1 => file address=address+f1-b1, otherwise,

b2<address<e2 => 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.

So that adb may be used on large files all appropriate values are kept as signed 32 bit integers.

## FILES

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

## 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.
When single stepping, system calls do not count as an executed instruction.
Local variables whose names are the same as an external variable may foul up the accessing of the external.

## NAME

admin - create and administer SCCS files

## SYNOPSIS

**admin [-n] [-i[name]] [-rrel] [-t[name]]**
**[-fflag[flag-val]] [-dflag[flag-val]]**
**[-alogin] [-elogin] [-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.

-rrel — The release 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.

-fflag — 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.

cceil — 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.

ffloor — The lowest release (i.e., ``floor''), a number greater than Ø 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.

dSID — 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.

**llist** 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.

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

**mmod** Module name of the SCCS file substituted for all occurrences of the %M% 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.

ttype Type of module in the SCCS file substituted for all occurrences of %Y% 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).

-dflag 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.

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

-alogin 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 logins, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.

-elogin 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.

## 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 loader. It can be used, though, for any similar purpose.

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 p, 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

**SEE ALSO**

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

**BUGS**

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

## NAME

as - assembler

## SYNOPSIS

as [ - ] [ -o objfile ] file ...

## DESCRIPTION

As assembles the concatenation of the named files. If the optional first argument - is used, all undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file objfile; if that is omitted, a.out is used. It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

## FILES

| | |
|---|---|
| /lib/as2 | pass 2 of the assembler |
| /tmp/atm[1-3]? | temporary |
| a.out | object |

## SEE ALSO

ld(1), nm(1), adb(1), a.out(5)

## DIAGNOSTICS

When an input file cannot be read, its name followed by a question mark is typed and assembly ceases. When syntactic or semantic errors occur, a single-character diagnostic is typed out together with the line number and the file name in which it occurred. Errors in pass 1 cause cancellation of pass 2. The possible errors are:

| | |
|---|---|
| ) | Parentheses error |
| ] | Parentheses error |
| < | String not terminated properly |
| * | Indirection used illegally |
| a | Error in address |
| b | Branch instruction is odd or too remote |
| e | Error in expression |
| f | Error in local (`f' or `b') type symbol |
| g | Garbage (unknown) character |
| i | End of file inside an if |
| m | Multiply defined symbol as label |
| o | Word quantity assembled at odd address |
| p | `.' different in pass 1 and 2 |
| r | Relocation error |
| u | Undefined symbol |
| x | Syntax error |

## BUGS

Syntax errors can cause incorrect line numbers in following diagnostics.

## NAME

at - execute commands at a later time

## SYNOPSIS

**at** time [ day ] [ file ]

## DESCRIPTION

At squirrels away a copy of the named file (standard input default) to be used as input to sh(1) at a specified later time. A cd(1) command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the user and group ID of the creator of the copy file.

The time is 1 to 4 digits, with an optional following `A', `P', `N' or `M' for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional day is either (1) a month name followed by a day number, or (2) a day of the week; if the word `week' follows invocation is moved seven days further off. Names of months and days may be recognizably truncated. Examples of legitimate commands are

```
at 8am jan 24
at 1530 fr week
```

At programs are executed by periodic execution of the command /usr/lib/atrun from cron(8). The granularity of at depends upon how often atrun is executed.

Standard output or error output is lost unless redirected.

## FILES

/usr/spool/at/yy.ddd.hhhh.uu
activity to be performed at hour hhhh of year day ddd of year yy. uu is a unique number.
/usr/spool/at/lasttimedone contains hhhh for last hour of activity.
/usr/spool/at/past directory of activities now in progress
/usr/lib/atrun program that executes activities that are due
pwd(1)

## SEE ALSO

calendar(1), cron(8)

## DIAGNOSTICS

Complains about various syntax errors and times out of range.

**BUGS**

Due to the granularity of the execution of /usr/lib/atrun, there may be bugs in scheduling things almost exactly 24 hours into the future.

## NAME

basename - strip filename affixes

## SYNOPSIS

**basename** string [ suffix ]

## DESCRIPTION

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

This shell procedure invoked with the argument /usr/src/cmd/cat.c compiles the named file and moves the output to cat in the current directory:

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

## SEE ALSO

sh(1)

**NAME**

bc - arbitrary-precision arithmetic language

**SYNOPSIS**

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

**DESCRIPTION**

Bc is an interactive processor for a language which 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
```

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to scale influences the number of digits to be retained on arithmetic operations in the manner of dc(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.

For 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.

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.

**FILES**

/usr/lib/lib.b mathematical library
dc(1) desk calculator proper

**SEE ALSO**

dc(1)
L. L. Cherry and R. Morris, BC - An arbitrary precision desk-calculator language

**BUGS**

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

**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

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**

cat - catenate and print

**SYNOPSIS**

cat [ -u ] 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 file is given, or if the argument `-' is encountered, cat reads from the standard input. Output is buffered in 512-byte blocks unless the standard output is a terminal or the -u option is present.

**SEE ALSO**

pr(1), cp(1)

**BUGS**

Beware of `cat a b >a' and `cat a b >b', which destroy input files before reading them.

**NAME**

cc, pcc - C compiler

**SYNOPSIS**

cc [ option ] ... file ...

pcc [ option ] ... file ...

**DESCRIPTION**

Cc is the UNIX C compiler. It accepts 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. See ld(1) for load-time options.

-c Suppress the loading 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 which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls monitor(3) 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).

-f In systems without hardware floating-point, use a version of the C compiler which handles floating-point constants and loads the object program with the floating-point interpreter. Do not use if the hardware is present.

-O Invoke an object-code optimizer.

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

-P Run only the macro preprocessor and place the result

for each `.c' file in a corresponding `.i' file and has no `#' lines in it.

**-E** Run only the macro preprocessor and send the result to the standard output. The output is intended for compiler debugging; it is unacceptable as input to cc.

**-o** *output*
Name the final output file *output*. If this option is used the file `a.out' will be left undisturbed.

**-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* `#include' files whose names do not begin with `/' are always sought first in the directory of the *file* argument, then in directories named in -I options, then in directories on a standard list.

**-B***string*
Find substitute compiler passes in the files named *string* with the suffixes cpp, cØ, cl and c2. If *string* is empty, use a standard backup version.

**-t**[pØ12]
Find only the designated compiler passes in the files whose names are constructed by a -B option. In the absence of a -B option, the *string* is taken to be `/usr/c/'.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier cc run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name a.out.

The major purpose of the `portable C compiler', pcc, is to serve as a model on which to base other compilers. Pcc does not support options -f, -E, -B, and -t. It provides, in addition to the language of cc, unsigned char type data and initialized bit fields.

FILES

| | |
|---|---|
| file.c | input file |
| file.o | object file |

| | |
|---|---|
| a.out | loaded output |
| /tmp/ctm? | temporaries for cc |
| /lib/cpp | preprocessor |
| /lib/c[Ø1] | compiler for cc |
| /usr/c/oc[Ø12] | backup compiler for cc |
| /usr/c/ocpp | backup preprocessor |
| /lib/fc[Ø1] | floating-point compiler |
| /lib/c2 | optional optimizer |
| /lib/crtØ.o | runtime startoff |
| /lib/mcrtØ.o | startoff for profiling |
| /lib/fcrtØ.o | startoff for floating-point interpretation |
| /lib/libc.a | standard library, see intro(3) |
| /usr/include | standard directory for `#include' files |
| /tmp/pc* | temporaries for pcc |
| /usr/lib/ccom | compiler for pcc |

SEE ALSO

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

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader. Of these, the most mystifying are from the assembler, as(1), in particular `m', which means a multiply-defined external symbol (function or data).

BUGS

The -O optimizer was designed to work with cc; its use with pcc is suspect.

**NAME**

cc86 - C compiler for the Intel iAPX86

**SYNOPSIS**

cc86 [ option ] ... file ...

**DESCRIPTION**

Cc86 is the UNIX C compiler. It accepts 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 `.b86' substituted for `.c'. The `.b86' 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 `.a86' are taken to be assembly source programs and are assembled, producing a `.b86' file.

The following options are interpreted by cc86. See ld86(1) for load-time options.

-c Suppress the loading 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 which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls monitor(3) 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). (not yet implemented)

-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed `.a86'.

-P Run only the macro preprocessor and place the result for each `.c' file in a corresponding `.i' file and has no `#' lines in it.

-E Run only the macro preprocessor and send the result to the standard output. The output is intended for compiler debugging; it is unacceptable as input to cc86.

-o output

Name the final output file output. If this option is used the file `a.out' will be left undisturbed.

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

-Uname Remove any initial definition of name.

-Idir `#include' files whose names do not begin with `/' are always sought first in the directory of the file argument, then in directories named in -I options, then in directories on a standard list.

-Bstring
Find substitute compiler passes in the files named string with the suffixes cpp, c86 and c2. If string is empty, use a standard backup version.

-t[pØ12]
Find only the designated compiler passes in the files whose names are constructed by a -B option. In the absence of a -B option, the string is taken to be `/usr/c/'.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier cc86 run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name b.out.

FILES

| | |
|---|---|
| file.c | input file |
| file.b86 | object file |
| b.out | loaded output |
| /tmp/ctm? | temporaries for cc |
| /lib/cpp | preprocessor |
| /usr/bin/c86 | compiler for cc86 |
| /usr/lib/crtØ.b86 | runtime startoff |
| /usr/lib/mcrtØ.b86 | startoff for profiling |
| /usr/lib/86libc.a | standard library, see intro(3) |
| /usr/include | standard directory for `#include' files |

SEE ALSO

B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, 1978
D. M. Ritchie, C Reference Manual
monitor(3), prof(1), ld86(1), adb(1)

## DIAGNOSTICS

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

**NAME**

cdc - change the delta commentary of an SCCS delta

**SYNOPSIS**

**cdc -rSID [-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 SCCS IDentification (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 MRs 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 MRs, it is removed and changed into a ``comment'' line. A list of all deleted MRs 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"b178-12345 !b177-54321 b179-00001"
-ytrouble s.file
```

adds b178-12345 and b179-00001 to the MR list, removes b177-54321 from the MR list, and adds the comment trouble to

delta 1.6 of s.file.

```
cdc -r1.6 s.file
MRs? !b177-54321 b178-12345 b179-00001
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

cfnt - clear loaded font

## SYNOPSIS

**cfnt** fontnum [ window ]

## DESCRIPTION

Cfnt clears font fontnum from window window. The font must have already been loaded into the window using **lfnt**. Fontnum must be in the range Ø to 6 (7 is the default font and cannot be cleared). If window is not supplied it defaults to the window in which the command is being executed in.

## SEE ALSO

lfnt(1) lsfnt(1) sfnt(1)

## NAME

chgrp - change group

## SYNOPSIS

**chgrp** group file ...

## DESCRIPTION

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

The user invoking chgrp must belong to the specified group, or be the super-user.

## FILES

/etc/group

## SEE ALSO

chgrp(8), chown(2), passwd(5), group(5)

## NAME

chmod - change mode

## SYNOPSIS

**chmod** mode file ...

## DESCRIPTION

The mode of each named file is 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. If who is omitted, the default is a but the setting of the file creation mask (see umask(2)) is taken into account.

Op can be + to add permission to the file's mode, - to take away permission and = 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). Letters 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.

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

```
chmod o-w file
chmod +x file
```

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.

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

**SEE ALSO**

ls(1), chmod(2), chown (1), stat(2), umask(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 UID 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 GID or a group name found in the group-ID file.

Only the super-user can change owner or group, in order to simplify as yet unimplemented accounting procedures.

## FILES

/etc/passwd
/etc/group

## SEE ALSO

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

**NAME**

clear - clear terminal screen

**SYNOPSIS**

**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

**BUGS**

## 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

diff(1), comm(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 [-bfx]**

## DESCRIPTION

Col reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ESC-7 in ASCII) 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 and output resulting from use of the tbl(1) preprocessor.

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.

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

The control characters SO (ASCII code Ø17), and SI (Ø16) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

Col normally converts white space to tabs to shorten printing time. If the **-x** option is given, this conversion is suppressed.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (Ø33) followed by one of 789, SI, SO, and VT (Ø13). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

## SEE ALSO

troff(1), tbl(1), greek(1)

## BUGS

Can't back up more than 128 lines.
No more than 8ØØ characters, including backspaces, on a line.

**NAME**

comb - combine SCCS deltas

**SYNOPSIS**

**comb [-o] [-s] [-psid] [-clist] 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 SCCS IDentification 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, and produces a three column output: lines only in file1; lines only in file2; and lines in both files. The filename `-' means the standard input.

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

**SEE ALSO**

cmp(1), diff(1), uniq(1)

**NAME**

cp - copy

**SYNOPSIS**

cp file1 file2

cp file ... directory

**DESCRIPTION**

File1 is copied onto file2. The mode and owner of file2 are preserved if it already existed; the mode of the source file is used otherwise.

In the second form, one or more files are copied into the directory with their original file-names.

Cp refuses to copy a file onto itself.

**SEE ALSO**

cat(1), pr(1), mv(1)

## 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.

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).
- **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 directory header bytes. Used when reading pdp11

generated cpio tapes.

**EXAMPLES**

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

```
ls | cpio -o >/dev/mtØ

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

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

```
find . -cpio /dev/rmtØ
```

**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.

## NAME

cpp - the C language preprocessor

## SYNOPSIS

/lib/cpp [ option ... ] [ ifile [ ofile ] ]

## DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the cc(1) command. Thus the output of cpp is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, cpp and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of cpp other than in this framework is not suggested. The preferred way to invoke cpp is through the cc(1) command since the functionality of cpp may someday be moved elsewhere. See m4(1) for a general macro processor.

Cpp optionally accepts two file names as arguments. Ifile and ofile are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following options to cpp are recognized:

**-P** Preprocess the input without producing the line control information used by the next pass of the C compiler.

**-C** By default, cpp strips C-style comments. If the -C option is specified, all comments (except those found on cpp directive lines) are passed along.

**-Uname**
Remove any initial definition of name, where name is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

| | |
|---|---|
| operating system: | ibm, gcos, os, tss, unix |
| hardware: | interdata, pdp11, u370, u3b, vax |
| UNIX System variant: | RES, RT |

**-Dname**
**-Dname=def**
Define name as if by a #define directive. If no =def is given, name is defined as 1.

**-Idir**
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 ifile 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 ifile argument is not searched.

Two special names are understood by cpp. The name **__LINE__** is defined as the current line number (as a decimal integer) as known by cpp, and **__FILE__** is defined as the current file name (as a C string) as known by cpp. They can be used anywhere (including in macros) just as any other defined name.

All cpp directives start with lines begun by #. The directives are:

**#define** name token-string
Replace subsequent instances of name with token-string.

**#define** name( arg, ..., arg ) token-string
Notice that there can be no space between name and the (. Replace subsequent instances of name followed by a (, a list of comma separated tokens, and a ) by token-string where each occurrence of an arg in the token-string is replaced by the corresponding token in the comma separated list.

**#undef** name
Cause the definition of name (if any) to be forgotten from now on.

**#include** "filename"
**#include** <filename>
Include at this point the contents of filename (which will then be run through cpp). When the <filename> notation is used, filename is only searched for in the standard places. See the -I option above for more detail.

**#line** integer-constant "filename"
Causes cpp to generate line control information for the next pass of the C compiler. Integer-constant is the line number of the next line and filename is the file where it comes from. If "filename" is not given, the current file name is unchanged.

**#endif**
Ends a section of lines begun by a test directive (#if, #ifdef, or #ifndef). Each test directive must have a matching #endif.

**#ifdef** name
The lines following will appear in the output if and only if name has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

**#ifndef** name
The lines following will not appear in the output if and only if name has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

**#if** constant-expression
Lines following will appear in the output if and only if the constant-expression evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary -, !, and ~ operators are all legal in constant-expression. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in constant-expression in these two forms: **defined** ( name ) or **defined** name. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by cpp should be used in constant-expression. In particular, the **sizeof** operator is not available.

**#else**
Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output. And vice versa.

The test directives and the possible **#else** directives can be nested.

**FILES**
/usr/include standard directory for **#include** files

**SEE ALSO**
cc(1), m4(1).

**DIAGNOSTICS**
The error messages produced by cpp are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

**NOTES**
When newline characters were found in argument lists for macros to be expanded, previous versions of cpp put out the newlines as they were found and expanded. The current version of cpp replaces these newlines with blanks to alleviate problems that the previous versions had.

## 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(1M)).

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 in order:

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).

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.

## FILES

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

## SEE ALSO

init(1M), sh(1).

## DIAGNOSTICS

A history of all actions by cron are recorded in /usr/adm/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.

## 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 cleartext 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. No doubt 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

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, Bell Telephone Laboratories assumes no responsibility for their use by the recipient. Further, Bell Laboratories assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

## NAME

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

## SYNOPSIS

csh [ -cefinstvVxX ] [ arg ... ]

## DESCRIPTION

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

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

When a login shell terminates it executes commands from the file `.logout' in the users home directory.

### Lexical structure

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

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

When the shell's input is not a terminal, the character `#' introduces a comment which continues to the end of the input line. It 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. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by `;', and are then executed sequentially. A sequence of pipelines may be executed without waiting for it to terminate by following it with an `&'. Such a sequence is automatically prevented from being terminated by a hangup signal; the nohup command need not be used.

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

**Substitutions**

We now describe the various transformations the shell performs on the input in the order in which they occur.

**History substitutions**

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

History substitutions begin with the character `!' and may begin anywhere in the input stream if a history substitution is not already in progress. 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 will be described later.

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

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

For definiteness, consider the following output from the history command:

```
    9  write michael
```

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

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

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

To select words from an event we can follow the event specification by a `:' and a designator for the desired words. The words of 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 `$' |

The `:' separating the event specification from the word designator can be omitted if the argument selector begins with a `↑', `$', `*' `-' or `%'. After the optional word designator can be placed a sequence of modifiers, each 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. In any case it is an error for no word to be applicable.

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

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

A special abbreviation of a history reference occurs when the first 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 `}' if necessary to insulate it from the characters which follow. Thus, after `ls -ld ~paul' we might do `!{l}a' to do `ls -ld ~paula', while `!la' would look for a command starting `la'.

**Quotations with ' and "**

The quotation of strings by `'' and `"' can be used to prevent all or some of the remaining substitutions. Strings enclosed in `'' are prevented any further interpretation. Strings enclosed in `"' are yet 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 Substitution below) does a `"' quoted string yield parts of more than one word; `' quoted strings never do.

**Alias substitution**

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

Thus if the alias for `ls' is `ls -l' the command `ls /usr' would map to `ls -l /usr', the argument list here being undisturbed. Similarly if the alias for `lookup' was `grep !↑ /etc/passwd' then `lookup bill' would map to `grep bill /etc/passwd'.

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

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can `alias print 'pr \!* | lpr'' to make a command which 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.

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

Other operations treat variables numerically. The `@' 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 the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by `$' characters. This expansion can be prevented by preceding the `$' with a `\' except within `"'s where it **always** occurs, and within `'s 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. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to 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 variables value separated by blanks. When the `:q' modifier is applied to a substitution the variable will expand 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 reference a variable which is not set.

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

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

$name[selector]
${name[selector]}
: May be used to select only some of the words from the value of name. The selector is subjected to `$' substitution and may consist of a single number or two

numbers separated by a `-'. The first word of a variables value is numbered `1'. If the first number of a range is omitted it defaults to `1'. If the last member of a range is omitted it defaults to `$#name'. The selector `*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

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

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

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

$*
Equivalent to `$argv[*]'.

The modifiers `:h', `:t', `:r', `:q' and `:x' may be applied to the substitutions above as may `:gh', `:gt' and `:gr'. If braces `{' '}' appear in the command form then the modifiers must appear within the braces. **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, `Ø' if it is not.

$?Ø
Substitutes `1' if the current input filename is know, `Ø' if it is not.

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

**Command and filename substitution**

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of

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

**Command substitution**

Command substitution is indicated by a command enclosed in ``'. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. 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. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

**Filename substitution**

If a word contains any of the characters `*', `?', `[' or `{' or begins with the character `~', then that word is a candidate for filename substitution, also known as `globbing'. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. 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 is used to refer to home directories. Standing alone, i.e. `~' it expands to the invokers 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 their home directory; thus `~ken' might expand to `/usr/ken' and `~ken/chmach' to `/usr/ken/chmach'. If the character `~' is

followed by a character other than a letter or `/' or appears not at the beginning of a word, it is left undisturbed.

The metanotation `a{b,c,d}e' is a shorthand for `abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus `~source/sl/{oldls,ls}.c' expands to `/usr/source/sl/oldls.c /usr/source/sl/ls.c' whether or not these files exist without any chance of error if the home directory for `source' is `/usr/source'. Similarly `../{memo,*box}' might expand to `../memo ../box ../mbox'. (Note that `memo' was not sorted with the results of matching `*box'.) As a special case `{', `}' and `{}' are passed 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 which is identical to word. Word is not subjected to variable, filename or command substitution, and each input line is compared to word before any substitutions are done on this input line. Unless a quoting `\', `"', `'' or ``' appears in word variable and command substitution is performed on the intervening lines, allowing `\' to quote `$', `\' and ``'. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

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

  If the variable noclobber is set, then the file must not exist or be a character special file (e.g. a terminal or `/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the `!' forms can be used and suppress this check.

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

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

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

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

**Expressions**

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

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

Here the precedence increases to the right, `==' and `!=', `<=' `>=' `<' and `>', `<<' and `>>', `+' and `-', `*' `/' and `%' being, in groups, at the same level. The `==' and `!=' operators compare their arguments as strings, all others operate on numbers. Strings which begin with `Ø' are considered octal numbers. Null or missing arguments are considered `Ø'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the

same word; except when adjacent to components of expressions which are syntactically significant to the parser (`&' `|' `<' `>' `(' `)') they should be surrounded by spaces.

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

| | |
|---|---|
| r | read access |
| w | write access |
| x | execute access |
| e | existence |
| o | ownership |
| z | zero size |
| f | plain file |
| d | directory |

The specified 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 then all enquiries return false, i.e. `0'. Command executions succeed, returning true, i.e. `1', if the command exits with status 0, otherwise they fail, returning false, i.e. `0'. If more detailed status information is required then the command should be executed outside of an expression and the variable status examined.

### Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict 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 will succeed on non-seekable inputs.)

### Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

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

**alloc**
Shows the amount of dynamic core in use, broken down into used and free core, and address of the last location in the heap. With an argument shows each used and free block on the internal dynamic memory chain indicating its address, size, and whether it is used or free. This is a debugging command and may not work in production versions of the shell; it requires a modified version of the system memory allocator.

**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 shells 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 `../'), then each component of the variable cdpath is checked to see if it has a subdirectory name. Finally, if all else fails but name is a shell variable whose value begins with `/', then this is tried to see if it is a directory.

**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 shells 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. Otherwise the words are echoed, separated by spaces.

**else**
**end**
**endif**
**endsw**
See the description of the foreach, if, switch, and while statements below.

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

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

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

The builtin command continue may be used to continue the loop prematurely and the builtin command break to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with `?' before any statements in the loop are executed. 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 which wish to 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 specified line.

**history**
Displays the history event list.

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

place on commands in simple if statements apply.

**nohup**
**nohup command**
The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. On the Computer Center systems at UC Berkeley, this also submits the process. Unless the shell is running detached, nohup has no effect. All processes detached with ``&'' are automatically nohup'ed. (Thus, nohup is not really needed.)

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

In any case, if the shell is running 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 redirections occurs exactly once, even if count is Ø.

**set**
**set** name
**set** name=word
**set** name[index]=word
**set** name=(wordlist)
The first form of the command shows the value of all shell variables. Variables which have other than a

single word as value print as a parenthesized word list. The second form sets name to the null string. The third form sets name to the single word. The fourth form sets the 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, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the time variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

**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 ØØ2 giving all access to the group and read and execute access to others or Ø22 giving all access except no write access for users in the group or others.

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

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

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

**wait**
All child processes are waited for. It 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 may be used to terminate or continue the loop prematurely. (The while and end must appear alone on their input lines.) Prompting

occurs here the first time through the loop as for the foreach statement if the input is a terminal.

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

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

The operators `*=', `+=', etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of expr which 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, argv, child, home, path, prompt, shell and status are always set by the shell. Except for child and status this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

The shell copies the environment variable PATH into the variable path, and copies the value back into the environment whenever path is set. Thus is is not necessary to worry about its setting other than in the file .cshrc as inferior csh processes will import the definition of path from the environment. (It could be set once in the .login except that commands through net(1) would not see the definition.)

**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** Gives a list of alternate directories searched to find subdirectories in 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-builtin 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=",;"'' will cause the history characters to be comma and semicolon.

**history** Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. 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 which are terminals. This prevents shells from accidentally being killed by control-D's.

**mail** The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says `You have new mail.' if the file exists with an access time not greater than its modify time.

If the first word of the value of mail is numeric it specifies a different mail checking interval, in seconds, than 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 which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

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

**path** Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no path variable then only full path names will execute. 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 which is given neither the -c nor the -t option will normally hash the contents of the directories in the path variable after reading .cshrc, and each time the path variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the rehash or the commands may not be found.

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

**shell** The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of Non-builtin 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 Ø2ØØ is added to the status. Builtin commands which fail return exit status `1', all other builtin commands set status `Ø'.

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

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

**Non-builtin command execution**

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via exec(2). Each word in the variable path names a directory from which the shell will attempt to execute the command. If it is given neither a -c nor a -t option, the shell will hash the names in these directories into an internal table so that it will only try an exec in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via unhash), or if the shell was given a -c or -t argument, and in any case for each directory component of path which does not begin with a ``/'', the shell concatenates 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' prints the home directory; leaving you where you were (printing this after the home directory), while `cd ; pwd' 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 an a new shell is spawned to read it.

If there is an alias for shell then the words of the alias will be 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 Ø to the shell is `-' then this is a login shell. The flag arguments are interpreted as follows:

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

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

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

**-i** The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their 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 `\' may be used to escape the newline at the end of this line and continue onto another line.

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

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

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

**-X** 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 was 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 `$Ø'. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute 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. The interrupt and quit signals are ignored for an invoked command if the command is followed by `&'; otherwise the signals have the values which the shell inherited from its parent. The shells handling of interrupts can be controlled by onintr. Login shells catch the terminate signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file `.logout'.

**AUTHOR**

William Joy

**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. Early version 6 systems typically have 512 character limits while later version 6 and version 7 systems have 5120 character limits. The number of arguments to a command which involves filename expansion is limited to 1/6'th 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 substitutions on a single line to 20.

**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. All and 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. Oring in 0200 to status on abnormal termination is a kludge.

In order to be able to recover from failing exec commands on version 6 systems, the new 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.

There are a number of bugs 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 core dump.

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 which have been fixed in the process control version. This version is not supported.

## NAME

cut - cut out selected fields of each line of a file

## SYNOPSIS

**cut -clist [file1 file2 ...]**
**cut -flist [-dchar] [-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).

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

-flist — 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.

-dchar — 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

date - print and set the date

## SYNOPSIS

**date** [ yymmddhhmm [ .ss ] ]

## DESCRIPTION

If no argument is given, the current date and time are printed. If an argument is given, the current date is set. yy is the last two digits of the year; 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; .ss is optional and is the seconds. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The year, month and day may be omitted, the current values being the defaults. The system operates in GMT. Date takes care of the conversion to and from local standard and daylight time.

## FILES

/usr/adm/wtmp to record time-setting

## SEE ALSO

utmp(5)

## 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.

**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 Ø-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 and in registers 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 executed 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.

An example which prints the first ten values of n! is

```
[lal+dsa*plalØ>y]sy
Øsal
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= | input file name; standard input is default |
| of= | 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 copy need be done |
| cbs=n | conversion buffer size |
| skip=n | skip n input records before starting copy |
| files=n | copy n files from (tape) input |
| seek=n | seek n records from beginning of output file before copying |
| count=n | copy only n input records |
| 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 new-line 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.

For example, to 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.

To skip over a file before copying from magnetic tape do

```
(dd of=/dev/null; dd of=x) </dev/rmt0
```

**SEE ALSO**

cp(1), tr(1)

**DIAGNOSTICS**

f+p records in(out): numbers of full and partial records 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.

Newlines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

## NAME

delta - make a delta (change) to an SCCS file

## SYNOPSIS

**delta** [-rSID] [-s] [-n] [-glist] [-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.

-rSID — 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 gets 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).

-glist — 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 ØØ1) 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.

**NAME**

df - disk free

**SYNOPSIS**

df [ -i ] [ -l ] [ filesystem ... ] [ file ... ]

**DESCRIPTION**

Df prints out the amount of free disk space available on the specified filesystem, e.g. ``/dev/dkØa'', or on the filesystem in which the specified file, e.g. ``$HOME'', is contained. If no file system is specified, the free space on all of the normally mounted file systems is printed. The reported numbers are in kilobytes.

Other options are:

**-i** Report also the number of inodes which are used and free.

**-l** examines also the free list, double checking that the summary number in the filesystem superblock is correct.

**FILES**

/etc/fstab list of normally mounted filesystems

**SEE ALSO**

fstab(5), icheck(8), quot(8)

## NAME

diff - differential file and directory comparator

## SYNOPSIS

**diff [ -l ] [ -r ] [ -s ] [ -cefh ] [ -b ] dir1 dir2**
**diff [ -cefh ] [ -b ] file1 file2**
**diff [ -Dstring ] [ -b ] file1 file2**

## DESCRIPTION

If both arguments are directories, diff sorts the contents of the directories by name, and then runs the regular file diff algorithm (described below) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed. Options when comparing directories are:

**-l** long output format; each text file diff is piped through pr(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.

**-r** causes application of diff recursively to common subdirectories encountered.

**-s** causes diff to report files which are the same, which are otherwise not mentioned.

**-Sname**
starts a directory diff in the middle beginning with file name.

When run on regular files, and when comparing text files which differ during directory comparison, diff tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, diff finds a smallest sufficient set of file differences. If neither file1 nor file2 is a directory, then either may be given as '-', in which case the standard input is used. If file1 is a directory, then a file in that directory whose file-name is the same as the file-name of file2 is used (and vice versa).

There are several options for output format; the default output format 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 `>'.

Except for -b, which may be given with any of the others, the following options are mutually exclusive:

-e producing a script of a, c and d commands for the editor ed, which will recreate file2 from file1. 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
```

Extra commands are added to the output when comparing directories with -e, so that the result is a sh(1) script for converting text files which are common to the two directories from their state in dir1 to their state in dir2.

-f produces a script similar to that of -e, not useful with ed, and in the opposite order.

-c produces a diff with lines of context. The default is to present 3 lines of context and may be changed, e.g to 10, by -c10. With -c the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from file1 are marked with `-'; those added to file2 are marked `+'. Lines which are changed from one file to the other are marked in both files with `!'.

-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.

-Dstring causes diff to create a merged version of file1 and file2 on the standard output, with C preprocessor controls included so that a compilation of the result without defining string is equivalent to compiling file1, while defining string will yield file2.

-b causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equal.

FILES

/tmp/d?????
/usr/lib/diffh for -h
/usr/bin/pr

SEE ALSO

cmp(1), cc(1), comm(1), ed(1)

DIAGNOSTICS

Exit status is Ø for no differences, 1 for some, 2 for trouble.

BUGS

Editing scripts produced under the -e or -f option are naive about creating lines consisting of a single '.'.

When comparing directories with the -b option specified, diff first compares the files ala cmp, and then decides to run the diff algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string differences.

## 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/diff3

## SEE ALSO

diff(1)

## BUGS

Text lines that consist of a single `.' will defeat -e.

## NAME

du - summarize disk usage

## SYNOPSIS

du [ -s ] [ -a ] [ name ... ]

## DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each specified directory or file name. If name is missing, `.' is used.

The optional argument -s causes only the grand total 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.

A file which has two links to it is only counted once.

## BUGS

Non-directories given as arguments (not under -a option) are not listed.
If there are too many distinct linked files, du counts the excess files multiply.

## NAME

dump - incremental file system dump

## SYNOPSIS

**dump** [ key [ argument ... ] filesystem ]

## DESCRIPTION

Dump copies to magnetic tape all files changed after a certain date in the filesystem. 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 filesystem 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 filesystem 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 filesystem 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 2300 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 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 used every fourth, tape 3 used every eighth, etc.)

```
dump 9u
```

When the level 9 incremental approaches a full tape (about 78000 blocks at 1600 BPI blocked 20), a level 1 dump should be made.

```
dump 1u
```

After this, the exponential series should progress as 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 dump can be carried as far as desired.

**FILES**

default filesystem and tape vary with installation.
/etc/ddate: record dump dates of filesystem/level.

**SEE ALSO**

restor(1), dump(5), dumpdir(1)

**DIAGNOSTICS**

If the dump requires more than one tape, it will ask you to change tapes. Reply with a new-line when this has been done.

**BUGS**

Sizes are based on 1600 BPI blocked tape. The raw magtape device has to be used to approach these densities. Read errors on the filesystem are ignored. Write errors on the magtape are usually fatal.

**NAME**

echo - echo arguments

**SYNOPSIS**

**echo [ -n ] [ arg ] ...**

**DESCRIPTION**

Echo writes its arguments separated by blanks and terminated by a newline on the standard output. If the flag -n is used, no newline is added to the output.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, do `echo ... 1>&2'.

## NAME

ed - text editor

## SYNOPSIS

**ed [ - ] [ -x ] [ name ]**

## DESCRIPTION

Ed is the standard text editor.

If a name 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. If -x is present, an x command is simulated first to handle an encrypted file. The optional - suppresses the printing of character counts by e, r, and w commands.

Ed operates on a copy of any file it is editing; changes made in 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.

Commands to ed have a simple and regular structure: zero or more addresses followed by a single character command, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to 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. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression. In the following specification for regular expressions the word `character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus \[. and sometimes ^*$.

2. A . matches any character.

3. A \ followed by any character except a digit or () matches that character.

4. A nonempty string s bracketed [s] (or [^s]) matches any character in (or not in) s. In s, \ has no special

meaning, and ] may only appear as the first letter. A substring a-b, with a and b in ascending ASCII order, stands for the inclusive range of ASCII characters.

5. A regular expression of form 1-4 followed by * matches a sequence of Ø or more matches of the regular expression.

6. A regular expression, x, of form 1-8, bracketed \(x\) matches what x matches.

7. A \ followed by a digit n matches a copy of the string that the bracketed regular expression beginning with the nth \( matched.

8. A regular expression of form 1-8, x, followed by a regular expression of form 1-7, y matches a match for x followed by a match for y, with the x match being as long as possible while still permitting a y match.

9. A regular expression of form 1-8 preceded by ^ (or followed by $), is constrained to matches that begin at the left (or end at the right) end of a line.

1Ø. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.

11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see s below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by `\'. This also applies to the character bounding the regular expression (often `/') and to `\' itself.

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; however, the exact effect on the current line is discussed under the description of the 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 name x, which must be a lower-case letter. Lines are marked with the k command described below.

5. A regular expression enclosed in slashes `/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.

6. A regular expression enclosed in queries `?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.

7. An address followed by a plus sign `+' or a minus sign `-' followed by a decimal number specifies that address plus (resp. 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 (resp. subtracted). As a consequence of this rule and rule 8, the address `-' refers to the line before the current line. Moreover, trailing `+' and `-' characters have cumulative effect, so `--' refers to the current line less 2.

10. To maintain compatibility with earlier versions of the editor, the character `^' in addresses is equivalent to `-'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma `,'. They may also be separated by a semicolon `;'. In this case the current line `.' is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches (`/', `?'). The second address of any two-address sequence must correspond to a line following 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, but are used to show that the given addresses are the default.

As mentioned, it is generally illegal for more than one command to appear on a line. However, most commands may be suffixed by `p' or by `l', in which case the current line is either printed or listed respectively in the way discussed below.

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. `.' is left on the last line input, if there were any, otherwise at the addressed line. Address `Ø' is legal for this command; text is placed at the beginning of the buffer.

(., .)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. `.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(., .)d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit 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. The number of characters read is typed. `filename' is remembered for possible use as a default file name in a subsequent r or w command. If `filename' is missing, the remembered name is used.

E filename

This command is the same as e, except that no diagnostic results when no w has been given since the last buffer alteration.

f filename

The filename command prints the currently remembered file name. If `filename' is given, the currently

remembered file name is changed to `filename'.

(1,$)g/regular expression/command list
In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with `.' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with `\'. A, i, and c commands and associated input are permitted; the `.' terminating input mode may be omitted if it would be on the last line of the command list. The commands g and v are not permitted in the command list.

(.)i

<text>
.

This command inserts the given text before the addressed line. `.' is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the a command only in the placement of the text.

(., .+1)j
This command joins the addressed lines into a single line; intermediate newlines simply disappear. `.' is left at the resulting line.

( . )kx
The mark command marks the addressed line with name x, which must be a lower-case letter. The address form `'x' then addresses this line.

(., .)l
The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The l command may be placed on the same line after any non-i/o command.

(., .)ma
The move command repositions the addressed lines after the line addressed by a. The last of the moved lines becomes the current line.

(., .)p
The print command prints the addressed lines. `.' is left at the last line printed. The p command may be placed on the same line after any non-i/o command.

(., .)P
This command is a synonym for p.

q The quit command causes ed to exit. No automatic write of a file is done.

Q This command is the same as q, except that no diagnostic results when no w has been given since the last buffer alteration.

($)r filename
The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see e and f commands). The file name is remembered if there was no remembered file name already. Address `Ø' 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 left at the last line read in from the file.

( ., .)s/regular expression/replacement/ or,
( ., .)s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, 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 regular expression and the replacement. `.' is left at the last line substituted.

An ampersand `&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of `&' in this context may be suppressed by preceding it by `\'. The characters `\n' where n is a digit, are replaced by the text matched by the n-th regular subexpression enclosed between `\(' and `\)'. When nested, parenthesized subexpressions are present, n is determined by counting occurrences of `\(' starting from the left.

Lines may be split by substituting new-line characters into them. The new-line in the replacement string must be escaped by preceding it by `\'.

(., .)ta
This command acts just like the m command, except that

a copy of the addressed lines is placed after address a (which may be Ø). `.' is left on the last line of the copy.

(., .)u
The undo command restores the preceding contents of the current line, which must be the last line in which a substitution was made.

(1, $)v/regular expression/command list
This command is the same as the global command g except that the command list is executed g with `.' initially set to every line except those matching the regular expression.

(1, $)w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writable by everyone). The file name is remembered if there was no remembered file name already. If no file name is given, the 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 printed.

(1,$)W filename
This command is the same as w, except that the addressed lines are appended to the file.

x A key string is demanded from the standard input. Later r, e 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 sh(1) to be interpreted as a command. `.' is unchanged.

(.+1)<newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to `.+1p'; it is useful for stepping through text.

If an interrupt signal (ASCII DEL) 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 temporary file. The limit on the number of lines depends on the amount of core: each line takes 1 word.

When reading a file, ed discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.

**FILES**

/tmp/e*
ed.hup: work is saved here if terminal hangs up

**SEE ALSO**

B. W. Kernighan, A Tutorial Introduction to the ED Text Editor
B. W. Kernighan, Advanced editing on UNIX
sed(1), crypt(1)

**DIAGNOSTICS**

`?name' for inaccessible file; `?' for errors in commands; `?TMP' for temporary file overflow.

To protect against throwing away valuable work, a q or e command is considered to be in error, unless a w has occurred since the last buffer change. A second q or e will be obeyed regardless.

**BUGS**

The l command mishandles DEL.
A ! command cannot be subject to a g command.
Because Ø is an illegal address for a 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

**edit [ -r ] name ...**

## DESCRIPTION

Edit is a variant of the text editor ex(1) 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 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 which you can then edit, and tells you how many lines and characters are in the file. 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 ``1Ø,2Ømove $'' for example. It is not necessary to use named buffers in this case (but you can if you wish).

**SEE ALSO**

ex(1), vi(1), `Edit: A tutorial', by Ricki Blau and James Joyce

**AUTHOR**

William Joy

**BUGS**

See ex(1).

**NAME**

egrep - search a file for a pattern

**SEE**

grep(1)

## NAME

ex - text editor

## SYNOPSIS

**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 will find that the editor edit is 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.Ø. 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.Ø 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 of the capabilities of terminals than ed does, and 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/Exnnnnn | editor temporary |
| /tmp/Rxnnnnn | named buffer temporary |
| /usr/preserve | preservation directory |

**SEE ALSO**

awk(1), ed(1), grep(1), sed(1), edit(1), grep(1), termcap(1), vi(1)

**AUTHOR**

William Joy

**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** arg ...

**DESCRIPTION**

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expr | expr
yields the first expr if it is neither null nor `Ø', otherwise yields the second expr.

expr & expr
yields the first expr if neither expr is null or `Ø', otherwise yields `Ø'.

expr relop expr
where relop is one of < <= = != >= >, yields `1' if the indicated comparison is true, `Ø' if false. The comparison is numeric if both expr are integers, otherwise lexicographic.

expr + expr
expr - expr
addition or subtraction of the arguments.

expr * expr
expr / expr
expr % expr
multiplication, division, or remainder of the arguments.

expr : expr
The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of ed(1). The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched (`Ø' on failure).

( expr )
parentheses for grouping.

Examples:

To add 1 to the Shell variable a:

```
a=`expr $a + 1`
```

To find the filename part (least significant part) of the pathname stored in variable a, which may or may not contain `/':

```
expr $a : '.*/\(.*\)' '|' $a
```

Note the quoted Shell metacharacters.

SEE ALSO

ed(1), sh(1), test(1)

DIAGNOSTICS

Expr returns the following exit codes:

| | |
|---|---|
| Ø | if the expression is neither null nor `Ø', |
| 1 | if the expression is null or `Ø', |
| 2 | for invalid expressions. |

NAME

true, false - provide truth values

SYNOPSIS

true

false

DESCRIPTION

True and false are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

EXAMPLE

```
while false
do
     command list
done
```

SEE ALSO

csh(1), sh(1), true(1)

DIAGNOSTICS

False has exit status nonzero.

**NAME**

fgrep - search a file for a pattern

**SEE**

grep(1)

**NAME**

f77 - Fortran 77 compiler

**SYNOPSIS**

f77 [ option ] ... file ...

**DESCRIPTION**

F77 is the UNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with `.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with `.o' substituted for '.f'.

Arguments whose names end with `.r' are taken to be Ratfor source programs; these are first transformed by the preprocessor, then compiled by f77.

In the same way, arguments whose names end with `.c' or `.s' are taken to be C or assembly source programs and are compiled or assembled, producing a `.o' file.

The following options have the same meaning as in cc(1). See ld(1) for load-time options.

-c Suppress loading and produce `.o' files for each source file.

-w Suppress all warning messages. If the option is `-w66', only Fortran 66 compatibility warnings are suppressed.

-p Prepare object files for profiling, see prof(1).

Name the final output file output instead of `a.out'.

The following options are peculiar to f77.

**-onetrip**
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

-u Make the default type of a variable `undefined' rather than using the default Fortran rules.

-C Compile code to check that subscripts are within declared array bounds.

-F Apply Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to `.f', but

do not compile.

-m Apply the M4 preprocessor to each `.r' file before transforming it with the Ratfor preprocessor.

-Rx Use the string x as a Ratfor option in processing `.r' files.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name `a.out'.

FILES

| | |
|---|---|
| file.[frsc] | input file |
| file.o | object file |
| a.out | loaded output |
| /lib/f77pass1 | compiler |
| /lib/fort | pass 2 |
| /usr/lib/libF77.a | intrinsic function library |
| /usr/lib/libI77.a | Fortran I/O library |
| /lib/libc.a | C library, see section 3 |

SEE ALSO

S. I. Feldman, P. J. Weinberger, A Portable Fortran 77 Compiler
prof(1), cc(1), ld(1), ratfor(1)

DIAGNOSTICS

The diagnostics produced by f77 itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

BUGS

The Fortran 66 subset of the language has been exercised extensively; the newer features have not.

## NAME

file - determine file type

## SYNOPSIS

**file 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.

## BUGS

It often makes mistakes. In particular it often suggests that command files are C programs.

**NAME**

find - find files

**SYNOPSIS**

**find** pathname-list expression

**DESCRIPTION**

Find recursively descends the directory hierarchy for each pathname in the pathname-list (i.e., one or more pathnames) seeking files that match a boolean expression written in the primaries given below. 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** filename
True if the filename argument 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 (Ø17777, 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** or **f** for block special file, character special file, directory or plain file.

**-links** n True if the file has n links.

**-user** uname
True if the file belongs to the user uname (login name or numeric user ID).

**-group** gname
True if the file belongs to group gname (group name or numeric group ID).

**-size** n True if the file is n blocks long (512 bytes per block).

**-inum** n True if the file has inode number n.

**-atime** n True if the file has been accessed in n days.

**-mtime** n True if the file has been modified in n days.

**-exec** command
True if the executed command returns a zero value as exit status. The end of the command must be

punctuated by an escaped semicolon. A command argument `{}' is replaced by the current pathname.

-ok command
Like **-exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

-print Always true; causes the current pathname to be printed.

-newer file
True if the current file has been modified more recently than the argument file.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

2) The negation of a primary (`!' is the unary not operator).

3) Concatenation of primaries (the and operation is implied by the juxtaposition of two primaries).

4) 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**

sh(1), test(1), filsys(5)

**BUGS**

The syntax is painful.

## NAME

fsck, sfsck - file system consistency check and interactive repair

## SYNOPSIS

**/etc/fsck [ -y ] [ -n ] [ -sX ] [ -SX ] [ -t filename ] [** filesystem **] ...**

## 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.

**fsck** is a small version of fsck which will execute in systems with less memory but will run slower and may require a scratch file during its operation.

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.

-sX 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 -sX option allows for creating an optimal free-list organization. The following forms of X are supported for the following devices:

-sBlocks-per-cylinder:Blocks-to-skip (for anything

If X is not given, the values used when the filesystem was created are used. If these values were not specified, then the value 400:9 is used.

-SX Conditionally reconstruct the free list. This option is like -sX 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 filesystem being checked, and if it is not a special file or did not already exist, it is removed when fsck completes.

If no filesystems are given to fsck then a default list of file systems is read 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 filesystem 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.

**SEE ALSO**

checklist(5), fs(5), crash(8).

**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 should understand about quotas.

## NAME

get - get a version of an SCCS file

## SYNOPSIS

get [-rSID] [-ccutoff] [-ilist] [-xlist] [-aseq-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.

-rSID The SCCS IDentification 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.

-ccutoff 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, -c75Ø2 is equivalent to -c75Ø228235959. 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.

-ilist 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.

-xlist 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''.

center expand ; c s s s s cl cl cl cl cØ c c c c c l c l l l .
TABLE 1. Determination of SCCS Identification String

| = SID* -b Specified | 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 hR.mL** exist | hR.mL.(mB+1).1 | R |
| R | - | R does not exist in release > R | Trunk succ.# | R.mL |
| R.mL | 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. | 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

getty - set terminal type, modes, speed, and line discipline

## SYNOPSIS

**/etc/getty** [ **-h** ] [ **-t** timeout ] line [ speed [ type [ linedisc ] ] ]
**/etc/getty -c** file

## DESCRIPTION

Getty is a program that is invoked by init(1M). It is the second process in the series, (init-getty-login-shell) that ultimately connects a user with the UNIX System. Initially getty prints the login message field for the entry it is using from /etc/gettydefs. Getty reads the user's login name and invokes the login(1) command with the user's name as argument. While reading the name, getty attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in /dev to which getty is to attach itself. Getty uses this string as the name of a file in the /dev directory to open for reading and writing. Unless getty is invoked with the -h flag, getty will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The -t flag plus timeout in seconds, specifies that getty should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, speed, is a label to a speed and tty definition in the file /etc/gettydefs. This definition tells getty what speed to initially run at, what the login message should look like, what the inital tty settings are, and what speed to try next should the user indicate that the speed is inappropriate. (By typing a <break> character.) The default speed is 300 baud. The optional third argument, type, is a character string describing to getty what type of terminal is connected to the line in question. Getty understands the following types:

| | |
|---|---|
| **none** | default |
| **vt61** | DEC vt61 |
| **vt100** | DEC vt100 |
| **hp45** | Hewlett-Packard HP45 |
| **c100** | Concept 100 |

The default terminal is **nonep**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, linedisc, is a character string describing which

line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISCØ**.

When given no optional arguments, getty sets the speed of the interface to 3ØØ baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, newline characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the ``break'' key. This will cause getty to attempt the next speed in the series. The series that getty tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see ioctl(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, login is called with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to login, which will place them in the environment (see login(1)).

A check option is provided. When getty is invoked with the **-c** option and file, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See ioctl(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs

SEE ALSO

ct(1C), init(1M), login(1), ioctl(2), gettydefs(4), inittab(4), tty(7).

## NAME

graph - draw a graph

## SYNOPSIS

graph [ option ] ...

## 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 plot(1) filters.

If the coordinates of a point are followed by a nonnumeric 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 newlines.

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 Ø 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, Ø 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: Ø disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

**-s** Save screen, don't erase before plotting.

**-x** [ 1 ]
If 1 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** [ 1 ]

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**

spline(1), plot(1)

**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**

grep, egrep, fgrep - search a file for a pattern

**SYNOPSIS**

**grep** [ option ] ... expression [ file ] ...

**egrep** [ option ] ... [ expression ] [ file ] ...

**fgrep** [ option ] ... [ strings ] [ file ]

**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; unless the -h flag is used, the file name is shown if there is more than one input file.

Grep patterns are limited regular expressions in the style of ed(1); it uses a compact nondeterministic algorithm. Egrep patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. Fgrep patterns are fixed strings; it is fast and compact.

The following options are recognized.

**-v** All lines but those matching are printed.

**-c** Only a count of matching lines is printed.

**-l** The names of files with matching lines are listed (once) separated by newlines.

**-n** Each line is preceded by its 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** No output is produced, only status.

**-h** Do not print filename headers with output lines.

**-y** Lower case letters in the pattern will also match upper case letters in the input (grep only).

**-e** expression
Same as a simple expression argument, but useful when the expression begins with a -.

**-f** file
The regular expression (egrep) or string list (fgrep)

is taken from the file.

-x (Exact) only lines matched in their entirety are printed (fgrep only).

Care should be taken when using the characters $ * [ ^ | ? ' " ( ) and \ in the expression as 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 (newline-separated) strings.

Egrep accepts extended regular expressions. In the following description `character' excludes newline:

A \ followed by a single character matches that character.

The character ^ ($) matches the beginning (end) of a line.

A . matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the string. Ranges of ASCII character codes may be abbreviated as in `a-zØ-9'. A ] may occur only as the first character of the string. A literal - must be placed where it can't be mistaken as a range indicator.

A regular expression followed by * (+, ?) matches a sequence of Ø or more (1 or more, Ø or 1) matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is [] then *+? then concatenation then | and newline.

SEE ALSO
ed(1), sed(1), sh(1)

## DIAGNOSTICS

Exit status is Ø 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.

## NAME

head - give first few lines of a stream

## SYNOPSIS

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.

## SEE ALSO

tail(1)

## AUTHOR

Bill Joy

**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

init, telinit - process control initialization

## SYNOPSIS

**/etc/init [Ø123456SsQq]**

**/etc/telinit [Ø123456sSQqabc]**

## DESCRIPTION

### Init

Init is a general process spawner. Its primary role is to create processes from a script stored in the file **/etc/inittab** (see inittab(4)). This file usually has init spawn getty's on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a run-level at any given time. A run-level can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by init for each of these run-levels is defined in the inittab file. Init can be in one of eight run-levels, **Ø-6** and **S** or **s**. The run-level is changed by having a privileged user run **/etc/init** (which is linked to /etc/telinit). This user spawned init sends appropriate signals to the orginal init spawned by the operating system when the system was rebooted, telling it which run-level to change to.

Init is invoked inside the UNIX System as the last step in the boot procedure. The first thing init does is to look for **/etc/inittab** and see if there is an entry of the type initdefault (see inittab(4)). If there is, init uses the run-level specified in that entry as the initial run-level to enter. If this entry is not in inittab or inittab is not found, init requests that the user enter a run-level from the virtual system console, **/dev/syscon**. If an **S (s)** is entered, init goes into the SINGLE USER level. This is the only run-level that doesn't require the existence of a properly formatted inittab file. If **/etc/inittab** doesn't exist, then by default the only legal run-level that init can enter is the SINGLE USER level. In the SINGLE USER level the virtual console terminal **/dev/syscon** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the SINGLE USER run-level one of two options can be elected. First, if the shell is terminated (via an end-of-file), init will reprompt for a new run-level. Second, the init or telinit command can signal init and force it to change the run-level of the system.

When attempting to boot the system, failure of init to

prompt for a new run-level may be due to the fact that the device /dev/syscon is linked to a device other than the physical system teletype (/dev/systty). If this occurs, init can be forced to relink /dev/syscon by typing a delete on the system teletype which is co-located with the processor.

When init prompts for the new run-level, the operator may only enter one of the digits 0 through 6 or the letters S or s. If S is entered init operates as previously described in SINGLE USER mode with the additional result that /dev/syscon is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the physical console, /dev/systty, saying where the virtual terminal has been relocated.

When init comes up initially and whenever it switches out of SINGLE USER state to normal run states, it sets the ioctl(2) states of the virtual console, /dev/syscon, to those modes saved in the file /etc/ioctl.syscon. This file is written by init whenever SINGLE USER mode is entered. If this file doesn't exist when init wants to read it, a warning is printed and default settings are assumed.

If a 0 through 6 is entered init enters the corresponding run-level. Any other input will be rejected and the user will be re-prompted. If this is the first time init has entered a run-level other than SINGLE USER, init first scans inittab for special entries of the type boot and bootwait. These entries are performed, providing the run-level entered matches that of the entry before any normal processing of inittab takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The inittab file is scanned to find all entries that are to be processed for that run-level.

Run-level 2 is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multi-user environment.

In a multi-user environment, the inittab file is usually set up so that init will create a process for each terminal on the system.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When init receives a child death signal, telling it that a process it spawned has died, it records the fact and the reason it died in /etc/utmp and /etc/wtmp if it exists (see who(1)). A history of the processes spawned is kept in /etc/wtmp if

such a file exists.

To spawn each process in the inittab file, init reads each entry and for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the inittab file, init waits for one of its descendant processes to die, a powerfail signal, or until init is signaled by init or telinit to change the system's run-level. When one of the above three conditions occurs, init re-examines the inittab file. New entries can be added to the inittab file at any time; however, init still waits for one of the above three conditions to occur. To provide for an instantaneous response the **init Q** or **init q** command can wake init to re-examine the inittab file.

If init receives a powerfail signal (SIGPWR) and is not in SINGLE USER mode, it scans inittab for special powerfail entries. These entries are invoked (if the run-levels permit) before any further processing takes place. In this way init can perform various cleanup and recording functions whenever the operating system experiences a power failure. It is important to note that the powerfail entries should not use devices that must first be initialized (e.g. dzb lines) after a power failure has occurred.

When init is requested to change run-levels (via telinit), init sends the warning signal (**SIGTERM**) to all processes that are undefined in the target run-level. Init waits 2Ø seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

**Telinit**

Telinit, which is linked to /etc/init, is used to direct the actions of init. It takes a one character argument and signals init via the kill system call to perform the appropriate action. The following arguments serve as directives to init.

**Ø-6** tells init to place the system in one of the run-levels **Ø-6**.

**a,b,c** tells init to process only those **/etc/inittab** file entries having the **a**, **b** or **c** run-level set.

**Q,q** tells init to re-examine the **/etc/inittab** file.

**s,S** tells init to enter the single user environment. When this level change is effected, the virtual system teletype, **/dev/syscon**, is changed to the terminal from

which the command was executed.

Telinit can only be run by someone who is super-user or a member of group sys.

**FILES**

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/syscon
/dev/systty

**SEE ALSO**

getty(1M), login(1), sh(1), who(1), kill(2), inittab(4), utmp(4).

**DIAGNOSTICS**

If init finds that it is continuously respawning an entry from /etc/inittab more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user init (telinit). This prevents init from eating up system resources when someone makes a typographical error in the inittab file or a program is removed that is referenced in the inittab.

**NAME**

iostat - report I/O statistics

**SYNOPSIS**

**iostat** [ option ] ... [ interval [ count ] ]

**DESCRIPTION**

Iostat delves into the system and reports certain statistics kept about input-output activity. Information is kept about up to three different disks (DKØ, SWAP, DK1) and about typewriters. For each disk, IO completions and number of words transferred are counted; for typewriters collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. The tally goes into one of four categories, depending on whether the system is executing in user mode, in `nice' (background) user mode, in system mode, or idle. From all these numbers and from the known transfer rates of the devices it is possible to determine information such as the degree of IO overlap and average seek times for each device.

The optional interval argument causes iostat to report once each interval seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional count argument restricts the number of reports.

With no option argument iostat reports for each disk the number of transfers per minute, the milliseconds per average seek, and the milliseconds per data transfer exclusive of seek time. It also gives the percentage of time the system has spend in each of the four categories mentioned above.

The following options are available:

**-t** Report the number of characters of terminal IO per second as well.

**-i** Report the percentage of time spend in each of the four categories mentioned above, the percentage of time each disk was active (seeking or transferring), the percentage of time any disk was active, and the percentage of time spent in `IO wait:' idle, but with a disk active.

**-s** Report the raw timing information: 32 numbers indicating the percentage of time spent in each of the possible configurations of 4 system states and 8 IO states (3 disks each active or not).

**-b** Report on the usage of IO buffers.

**FILES**

/dev/kmem, /unix

**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 newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

-an 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.

-jn m
Join on the mth field of file n. If n is missing, use the mth 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.

-tc Use character c as a separator (tab character). Every appearance of c in a line is significant.

**SEE ALSO**

sort(1), comm(1), awk(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, look and awk(1)

are wildly incongruous.

**NAME**

kill - terminate a process with extreme prejudice

**SYNOPSIS**

**kill** [ -signo ] processid ...

**DESCRIPTION**

Kill sends signal 15 (terminate) to the specified processes. If a signal number preceded by `-' is given as first argument, that signal is sent instead of terminate (see signal(2)). This will kill processes that do not catch the signal; in particular `kill -9 ...' is a sure kill.

By convention, if process number Ø is specified, all members in the process group (i.e. processes resulting from the current login) are signaled.

The killed processes must belong to the current user unless he is the super-user. To shut the system down and bring it up single user the super-user may use `kill -1 1'; see init(8).

The process number of an asynchronous process started with `&' is reported by the shell. Process numbers can also be found by using ps(1).

**SEE ALSO**

ps(1), kill(2), signal(2)

**NAME**

ld - loader

**SYNOPSIS**

**ld** [ option ] file ...

**DESCRIPTION**

Ld combines several object programs into one, resolves external references, and searches libraries. 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 only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

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, and the library has not been processed by ranlib(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named `__.SYMDEF', then it is understood to be a dictionary for the library such as produced by ranlib; the dictionary is searched iteratively to satisfy as many references as possible.

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 options. 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).

**-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**x This option is an abbreviation for the library name `/lib/libx.a', where x is a string. If that does not exist, ld tries `/usr/lib/libx.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(1) 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 4K word boundary following the end of the text.

**-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.

**-o** The name argument after -o is used as the name of the ld output file, instead of **a.out**.

**-e** The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

**-O** This is an overlay file, only the text segment will be replaced by exec(2). Shared data must have the same layout as in the program overlaid.

**-D** The next argument is a decimal number that sets the size of the data segment.

FILES

| | |
|---|---|
| /lib/lib*.a | libraries |
| /usr/lib/lib*.a | more libraries |
| a.out | output file |

**SEE ALSO**

as(1), ar(1), cc(1), ranlib(1)

**BUGS**

**NAME**

ld86 - loader for the Intel iAPX86

**SYNOPSIS**

**ld86** [ option ] file ...

**DESCRIPTION**

Ld86 combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object files are given, and ld68 combines them, producing an object module which can be either executed or become the input for a further ld86 run. (In the latter case, the **-r** option must be given to preserve the relocation bits.) The output of ld86 is left on **b.out**.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

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 may be 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.

Ld86 understands several options. 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).

**-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**x This option is an abbreviation for the library name `/usr/lib/86libx.a', where x is a string. If that does not exist, ld86 tries `/lib/86libx.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 cc86(1) 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 ld86 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.

**-o** The name argument after -o is used as the name of the ld86 output file, instead of b.out.

**-T** The next argument is a hex number giving the address at which the program is to be loaded into iSBC86/12 memory. A value of 4ØØ is typically used to load the program above reserved low memory data areas.

**FILES**

| | |
|---|---|
| /lib/86lib*.a | libraries |
| /usr/lib/86lib*.a | more libraries |
| b.out | output file |

**SEE ALSO**

a86(1), ar(1), cc86(1)

**BUGS**

**NAME**

lex - generator of lexical analysis programs

**SYNOPSIS**

**lex [ -tvfn ] [ file ] ...**

**DESCRIPTION**

Lex generates programs to be used in simple lexical analyis of text. The input files (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, `lex.yy.c' is generated, to be compiled thus:

```
cc lex.yy.c -ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The following lex program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A-Z] putchar(yytext[0]+'a'-'A');
[ ]+$
[ ]+  putchar(' ');
```

The options have the following meanings.

**-t** Place the result on the standard output instead of in file `lex.yy.c'.

**-v** Print a one-line summary of statistics of the generated analyzer.

**-n** Opposite of -v; -n is default.

**-f** `Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

**SEE ALSO**

yacc(1)

M. E. Lesk and E. Schmidt, LEX - Lexical Analyzer Generator

**NAME**

lfnt - load font

**SYNOPSIS**

**lfnt** [ fontnum ] [ fontname ] [ window ]

**DESCRIPTION**

Lfnt loads the font in file fontname into window window and assigns it number fontnum. If window is not supplied, it defaults to the current window. Fontname must be a complete pathname of the font file. Fontnum must be in the range Ø to 6 (7 is the default font and is automatically loaded).

**FILES**

/Fonts/CRT contains available fonts.

**SEE ALSO**

lsfnt(1) sfnt(1) cfnt(1)

## 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.

/*VARARGSn*/
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**

ln - make a link

**SYNOPSIS**

ln name1 [ name2 ]

**DESCRIPTION**

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There is no way to distinguish a link to a file from its original directory entry; any changes in the file are effective independently of the name by which the file is known.

Ln creates a link to an existing file name1. If name2 is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of name1.

It is forbidden to link to a directory or to link across file systems.

**SEE ALSO**

rm(1)

## NAME

login - sign on

## SYNOPSIS

**login [ username ]**

## DESCRIPTION

The login command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See `How to Get Started' for how to dial up initially.

If login is invoked without an argument, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of .mail and message-of-the-day files. Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually sh(1)) according to specifications found in a password file. Argument Ø of the command interpreter is `-sh.

Login is recognized by sh(1) and executed directly (without forking).

## FILES

| | |
|---|---|
| /etc/utmp | accounting |
| /usr/adm/wtmp | accounting |
| /usr/mail/* | mail |
| /etc/motd | message-of-the-day |
| /etc/passwd | password file |

## SEE ALSO

init(8), newgrp(1), getty(8), mail(1), passwd(1), passwd(5)

## DIAGNOSTICS

`Login incorrect,' if the name or the password is bad.
`No Shell', `cannot open password file', `no directory': consult a programming counselor.

**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.

**FILES**

| | |
|---|---|
| /etc/passwd | user's identification and accounting data. |
| /usr/lib/lpd | line printer daemon. |
| /usr/spool/lpd/* | spool area. |

**NAME**

ls - list contents of directory

**SYNOPSIS**

**ls [ -abcdfgilmqrstuxlCFR ]** name ...
**l [** ls options **]** name ...

**DESCRIPTION**

For each directory argument, ls lists the contents of the directory; for each file argument, ls repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

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 which 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; when invoked as l this format is also used.

There are an unbelievable number of 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 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 notation, 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 `ls -l'.
/etc/group to get group ID's for `ls -g'.

## 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 as ``ls -s'' is much different than ``ls -s | lpr''. On the other hand, not doing this setting would make old shell scripts which used ls almost certain losers.

Column widths choices are poor for terminals which can tab.

**NAME**

lsfnt - list loaded fonts

**SYNOPSIS**

**lsfnt**

**DESCRIPTION**

Lsfnt Displays a list of all loaded fonts for the window. The font number is listed (* next to the font currently selected) along with the pathname of the loaded font. The pathname is displayed in its own font type.

**SEE ALSO**

lfnt(1) sfnt(1) cfnt(1)

## NAME

m4 - macro processor

## SYNOPSIS

**m4 [ 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 arguments, or if an argument is `-', the standard input is read. The processed text is written on the standard output.

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 newlines 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 Ø is the name of the macro; missing arguments are replaced by the null string.

undefine — removes the definition of the macro named in its argument.

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.

changequote
Change quote characters to the first and second arguments. Changequote without arguments restores the original values (i.e., ``'').

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 newline.

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.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation); relationals; parentheses.

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.

maketemp fills in a string of XXXXX in its argument with the current process id.

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.

SEE ALSO

B. W. Kernighan and D. M. Ritchie, The M4 Macro Processor

## NAME

mail - send or receive mail among users

## SYNOPSIS

**mail person ...**
**mail [ -r ] [ -q ] [ -p ] [ -f file ]**

## DESCRIPTION

Mail with no argument prints a user's mail, message-by-message, in last-in, first-out order; the optional argument **-r** causes first-in, first-out order. If the **-p** flag is given, the mail is printed with no questions asked; otherwise, for each message, mail reads a line from the standard input to direct disposition of the message.

newline
Go on to next message.

d Delete message and go on to the next.

p Print message again.

\- Go back to previous message.

s [ file ] ...
Save the message in the named files (`mbox' default).

w [ file ] ...
Save the message, without a header, in the named files (`mbox' default).

m [ person ] ...
Mail the message to the named persons (yourself is default).

EOT (control-D)
Put unexamined mail back in the mailbox and stop.

q Same as EOT.

x Exit, without changing the mailbox file.

!command
Escape to the Shell to do command.

? Print a command summary.

An interrupt stops the printing of the current letter. The optional argument **-q** causes mail to exit after interrupts without changing the mailbox.

When persons are named, mail takes the standard input up to

an end-of-file (or a line with just `.') and adds it to each person's `mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with `>'. A person is usually a user name recognized by login(1). To denote a recipient on a remote system, prefix person by the system name and exclamation mark (see uucp(1)).

The -f option causes the named file, e.g. `mbox', to be printed as if it were the mail file.

Each user owns his own mailbox, which is by default generally readable but not writable. The command does not delete an empty mailbox nor change its mode, so a user may make it unreadable if desired.

When a user logs in he is informed of the presence of mail.

**FILES**

```
/usr/spool/mail/*   mailboxes
/etc/passwd    to identify sender and locate persons
mbox       saved mail
/tmp/ma*   temp file
dead.letter    unmailable text
uux(1)
```

**SEE ALSO**

xsend(1), write(1), uucp(1)

**BUGS**

There is a locking mechanism intended to prevent two senders from accessing the same mailbox, but it is not perfect and races are possible.

**NAME**

make - maintain program groups

**SYNOPSIS**

**make [ -f makefile ] [ option ] ... file ...**

**DESCRIPTION**

Make executes commands in makefile to update one or more target names. Name is typically a program. If no -f option is present, `makefile' and `Makefile' are tried in order. If makefile is `-', the standard input is taken. More than one -f option may appear

Make updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target.

Sharp and newline surround comments.

The following makefile says that `pgm' depends on two files `a.o' and `b.o', and that they in turn depend on `.c' files and a common file `incl'.

```
pgm: a.o b.o
     cc a.o b.o -lm -o pgm
a.o: incl a.c
     cc -c a.c
b.o: incl b.c
     cc -c b.c
```

Makefile entries of the form

**string1 = string2**

are macro definitions. Subsequent appearances of $(string1) are replaced by string2. If string1 is a single character, the parentheses are optional.

Make infers prerequisites for files for which makefile gives no construction commands. For example, a `.c' file may be inferred as prerequisite for a `.o' file and be compiled to produce the `.o' file. Thus the preceding example can be done more briefly:

```
pgm: a.o b.o
```

```
        cc a.o b.o -lm -o pgm
    a.o b.o: incl
```

Prerequisites are inferred according to selected suffixes listed as the `prerequisites' for the special name `.SUFFIXES'; multiple lists accumulate; an empty list clears what came before. Order is significant; the first possible name for which both a file and a rule as described in the next paragraph exist is inferred. The default list is

```
.SUFFIXES: .out .o .c .e .r .f .y .l .s
```

The rule to create a file with suffix s2 that depends on a similarly named file with suffix s1 is specified as an entry for the `target' s1s2. In such an entry, the special macro $* stands for the target name with suffix deleted, $@ for the full target name, $< for the complete list of prerequisites, and $? for the list of prerequisites that are out of date. For example, a rule for making optimized `.o' files from `.c' files is

```
.c.o: ; cc -c -O -o $@ $*.c
```

Certain macros are used by the default inference rules to communicate optional arguments to any resulting compilations. In particular, `CFLAGS' is used for cc and f77(1) options, `LFLAGS' and `YFLAGS' for lex and yacc(1) options.

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target `.SILENT' is in makefile, or the first character of the command is `@'.

Commands returning nonzero status (see intro(1)) cause make to terminate unless the special target `.IGNORE' is in makefile or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target depends on the special name `.PRECIOUS'.

Other options:

**-i** Equivalent to the special entry `.IGNORE:'.

**-k** When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

**-n** Trace and print, but do not execute the commands needed to update the targets.

-t Touch, i.e. update the modified date of targets, without executing any commands.

-r Equivalent to an initial special entry `.SUFFIXES:' with no list.

-s Equivalent to the special entry `.SILENT:'.

**FILES**

makefile, Makefile

**SEE ALSO**

sh(1), touch(1)
S. I. Feldman Make - A Program for Maintaining Computer Programs

**BUGS**

Some commands return nonzero status inappropriately. Use -i to overcome the difficulty.
Commands that are directly executed by the shell, notably cd(1), are ineffectual across newlines in make.

## NAME

makekey - generate encryption key

## SYNOPSIS

**/usr/lib/makekey**

## DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the input key) can be arbitrary ASCII characters. The last two (the salt) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the output key.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 output key bits in the result.

Makekey is intended for programs that perform encryption (e.g., ed(1) and crypt(1)). Usually, its input and output will be pipes.

## SEE ALSO

crypt(1), ed(1), passwd(4).

## NAME

man - find manual information by keywords; print out the manual

## SYNOPSIS

**man -k keyword ...**
**man -f file ...**
**man [ - ] [ -t ] [ section ] title ...**

## DESCRIPTION

Man is a program which gives information from the programmers manual. It can be asked form one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. It can also provide on-line access to the sections of the printed manual.

When given the option -k and a set of keywords, man prints out a one line synopsis of each manual sections whose listing in the table of contents contains that keyword.

When given the option -f and a list of file names, man attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither -k nor -f is specified, man formats a specified set of manual pages. If a section specifier is given man looks in the that section of the manual for the given titles. Section is an arabic section number, i.e. 3, which may be followed by a single letter classifier, i.e. 1g indicating a graphics program in section 1. If section is omitted, man searches all sections of the manual, giving preference to commands over subroutines in system libraries, and printing the first section it finds, if any.

If the standard output is a teletype, or if the flag - is given, then man pipes its output through cat(1) with the option -s to crush out useless blank lines, ul(1) to create proper underlines for different terminals, and through more(1) to stop after each page on the screen. Hit a space to continue, a control-D to scroll 11 more lines when the output stops.

The -t flag causes man to arrange for the specified section to be troff'ed to a suitable raster output device; see vtroff(1).

## FILES

/usr/man/man?/*
/usr/man/cat?/*

## SEE ALSO

more(1), ul(1), whereis(1), catman(8)

**BUGS**

The manual is supposed to be reproducible either on the phototypesetter or on a typewriter. However, on a typewriter 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*
/dev

## SEE ALSO

write(1)

## DIAGNOSTICS

Exit status is Ø if messages are receivable, 1 if not, 2 on error.

## 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 Ø if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

## NAME

mkfs - construct a file system

## SYNOPSIS

/etc/mkfs special proto

## DESCRIPTION

Mkfs constructs a file system by writing on the special file special according to the directions found in the prototype file proto. 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, see bproc(8). 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 i-list. 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 pathname 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 $.

If the prototype file cannot be opened and its name consists of 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 proto interpreted as a decimal number. The number of i-nodes is calculated as a function of the

filsystem size. The boot program is left uninitialized.

A sample prototype specification follows:

```
/usr/mdec/uboot
4872 55
d--777 3 1
usr  d--777 3 1
     sh   ---755 3 1 /bin/sh
     ken  d--755 6 1
          $
     bØ   b--644 3 1 Ø Ø
     cØ   c--644 3 1 Ø Ø
     $
$
```

**SEE ALSO**

filsys(5), dir(5), bproc(8)

**BUGS**

There should be some way to specify links.

## NAME

mknod - build special file

## SYNOPSIS

**/etc/mknod** name [ c ] [ b ] major minor

## DESCRIPTION

Mknod makes a special file. The first argument is the name of the entry. 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).

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file conf.c.

## SEE ALSO

mknod(2)

## NAME

mkstr -- create an error message file by massaging C source

## SYNOPSIS

**mkstr** [ - ] messagefile prefix file ...

## DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as 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, 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 which can 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) al, 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 mkstred program.

**SEE ALSO**

lseek(2), xstr(1)

**AUTHORS**

Bill Joy and Charles Haley

**BUGS**

All the arguments except the name of the file to be processed are unnecessary.

**NAME**

more, page - file perusal filter for crt viewing

**SYNOPSIS**

**more** [ **-d** ] [ **-f** ] [ **-l** ] [ **-**n ] [ +linenumber ] [ +/pattern ] [ name ... ]

**page** [ **-d** ] [ **-f** ] [ **-l** ] [ **-**n ] [ +linenumber ] [ +/pattern ] [ name ... ]

**DESCRIPTION**

More is a filter which 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 which is the size (in lines) of the window which 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)

iz same as typing a space except that i, if present, becomes the new window size.

is skip i lines and print a screenful of lines

if 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.

in 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
```

AUTHOR

Eric Shienbrood

FILES

| | |
|---|---|
| /etc/termcap | Terminal data base |
| /usr/lib/more.help | Help file |

SEE ALSO

script(1)

## NAME

mount, umount - mount and dismount file system

## SYNOPSIS

**/etc/mount [ special name [ -r ] ]**

**/etc/umount special**

## DESCRIPTION

Mount announces to the system that a removable file system is present on the device special. The file name must exist already; it must be a directory (unless the root of the mounted file system is not a directory). It becomes the name of the newly mounted root. The optional last argument indicates that the file system is to be mounted read-only.

Umount announces to the system that the removable file system previously mounted on device special is to be removed.

These commands maintain a table of mounted devices. If invoked without an argument, mount prints the table.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

## FILES

/etc/mtab: mount table

## SEE ALSO

mount(2), mtab(5)

## BUGS

Mounting file systems full of garbage will crash the system. Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

## NAME

mv - move or rename files and directories

## SYNOPSIS

**mv** file1 file2

**mv** file ... directory

## DESCRIPTION

Mv moves (changes the name of) file1 to file2.

If file2 already exists, it is removed before file1 is moved. If file2 has a mode which forbids writing, mv prints the mode (see chmod(2)) and reads the standard input to obtain a line; if the line begins with y, the move takes place; if not, mv exits.

In the second form, one or more files are moved to the directory with their original file-names.

Mv refuses to move a file onto itself.

## SEE ALSO

cp(1), chmod(2)

## BUGS

If file1 and file2 lie on different file systems, mv must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Mv should take -f flag, like rm, to suppress the question if the target exists and is not writable.

**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.

A password is demanded if the group has a password and the user himself does not.

When most users log in, they are members of the group named 'other.' Newgrp is known to the shell, which executes it directly without a fork.

**FILES**

/etc/group, /etc/passwd

**SEE ALSO**

login(1), group(5)

**NAME**

nice, nohup - run a command at low priority

**SYNOPSIS**

**nice** [ -number ] command [ arguments ]

**nohup** command [ arguments ]

**DESCRIPTION**

Nice executes command with low scheduling priority. If the number argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default number is 10.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. `--10'.

Nohup executes command immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. Nohup should be invoked from the shell with `&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal.

**FILES**

nohup.out standard output and standard error file under nohup

**SEE ALSO**

nice(2)

**DIAGNOSTICS**

Nice returns the exit status of the subject command.

## NAME

nm - print name list

## SYNOPSIS

nm [ -gnopru ] [ 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), 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.

-p Don't sort; print in symbol-table order.

-r Sort in reverse order.

-u Print only undefined symbols.

## SEE ALSO

ar(1), ar(5), a.out(5)

## NAME

nm86 - print name list for b.out format files

## SYNOPSIS

**nm86 [ -gnopru ] [ file ... ]**

## DESCRIPTION

Nm86 prints the name list (symbol table) of each b.out format object file in the argument list. B.out format object files are produced by a86 and cc86, which produce code for the Intel iAPX86.

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), 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.

**-p** Don't sort; print in symbol-table order.

**-r** Sort in reverse order.

**-u** Print only undefined symbols.

**-x** Print values in hex instead of octal.

## SEE ALSO

ar(1), ar(5), b.out(5)

**NAME**

nroff, troff - text formatting and typesetting

**SYNOPSIS**

**nroff** [ option ] ... [ file ] ...

**DESCRIPTION**

Nroff formats text in the named files for typewriter-like devices. See also troff(1). The full capabilities of nroff and troff are described in the Nroff/Troff User's Manual.

If no file argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input.

The options, which may appear in any order so long as they appear before the files, are:

-olist Print only pages whose page numbers appear in the comma-separated list of numbers and ranges. 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.

-nN Number first generated page N.

-sN Stop every N pages. Nroff will halt prior to every N pages (default N=1) to allow paper loading or changing, and will resume upon receipt of a newline.

-mname Prepend the macro file /usr/lib/tmac/tmac.name to the input files.

-raN Set register a (one-character) to N.

-i Read standard input after the input files are exhausted.

-q Invoke the simultaneous input-output mode of the rd request.

-Tname Prepare output for specified terminal. Known names are 37 for the (default) Teletype Corporation 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, and 450 for the DASI-450 (Diablo Hyterm).

-e Produce equally-spaced words in adjusted lines, using full terminal resolution.

**-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.

## EXAMPLE

```
nroff -s4 -me filea
```

will nroff the named file using the **-me** macro package, stopping every 4 pages.

## FILES

| | |
|---|---|
| /usr/lib/suftab | suffix hyphenation tables |
| /tmp/ta* | temporary file |
| /usr/lib/tmac/tmac.* | standard macro files |
| /usr/lib/term/* | terminal driving tables for nroff |

## SEE ALSO

J. F. Ossanna, Nroff/Troff user's manual
B. W. Kernighan, A TROFF Tutorial
troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

**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 argument characters are:

**b** Interpret bytes in octal.

**c** Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\Ø, backspace=\b, formfeed=\f, newline=\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 `+'.

Dumping continues until end-of-file.

**SEE ALSO**

adb(1)

**NAME**

p - file perusal filter for crt viewing

**SEE**

more(1)

**NAME**

page - file perusal filter for crt viewing

**SEE**

more(1)

**NAME**

passwd - change login password

**SYNOPSIS**

**passwd [ name ]**

**DESCRIPTION**

This command changes (or installs) a password associated with the user name (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

**FILES**

/etc/passwd

**SEE ALSO**

login(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, Password Security: A Case History

## NAME

paste - merge same lines of several files or subsequent lines of one file

## SYNOPSIS

```
paste file1 file2 ...
paste -dlist file1 file2 ...
paste -s [-dlist] file1 file2 ...
```

## DESCRIPTION

In the first two forms, paste concatenates corresponding lines of the given input files file1, file2, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). 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 \Ø (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

pmda - post-mortem dump analyzer

## SYNOPSIS

**pmda** [ -cpkrlbf ] [ file ] [ addr ] [ length ]

## DESCRIPTION

Pmda is used to read the contents of a file generated by the monitor post-mortem dump utility **pmd**. File contains the dump image (the file name defaults to **pmdump** if not specified). If the dump was output to tape, the tape must be first copied to a disk file using **dd** with a blocksize of 8k. Addr specifies the starting address with the addressing unit defined by the options (defaults to zero if not specified). Length is the number of bytes from the starting address to dump (defaults to 256 if not specified).

**h** Display contents of dump file header.

**f** The dump file is specified by the next parameter.

**c** addr is a click number.

**p** addr is a physical address.

**k** addr is a kernel virtual address.

**r** byte reverse the display.

**l** display values as longs (4 bytes).

**b** display values as bytes.

**defaults:**

File is **pmdump**. Addr is absolute dump file offest in bytes if c, p, or k options are not specified. Values are displayed as words (2 bytes).

## SEE ALSO

dd(1), pmd(1) in SDU user's manual

**NAME**

pr - print file

**SYNOPSIS**

**pr [ option ] ... [ file ] ...**

**DESCRIPTION**

Pr produces a printed listing of one or more files. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, pr prints its standard input.

Options apply to all following files but may be reset between files:

-n Produce n-column output.

+n Begin printing with page n.

-h Take the next argument as a page header.

-wn For purposes of multi-column output, take the width of the page to be n characters instead of the default 72.

-ln Take the length of the page to be n lines instead of the default 66.

-t Do not print the 5-line header or the 5-line trailer normally supplied for each page.

-sc Separate columns by the single character c instead of by the appropriate amount of white space. A missing c is taken to be a tab.

-m Print all files simultaneously, each in one column,

Inter-terminal messages via write(1) are forbidden during a pr.

**FILES**

/dev/tty? to suspend messages.

SEE ALSO

cat(1)

DIAGNOSTICS

There are no diagnostics when pr is printing on a terminal.

## NAME

printenv - print out the environment

## SYNOPSIS

**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 Ø.

## SEE ALSO

sh(1), environ(5), csh(1)

## BUGS

**NAME**

prof - display profile data

**SYNOPSIS**

**prof** [ **-v** ] [ **-a** ] [ **-l** ] [ **-**low [ **-**high ] ] [ file ]

**DESCRIPTION**

Prof interprets the file mon.out produced by the monitor 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 plot(1) filters. The numbers 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**

monitor(3), profil(2), cc(1), plot(1)

**BUGS**

Beware of quantization errors.

## 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 SCCS IDentification (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.

**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:
b178-12345
b179-54321
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.

## NAME

ps - process status

## SYNOPSIS

**ps [ acgklrstuvwx# [ corefile ] [ swapfile ] [ system ] ]**

## DESCRIPTION

Ps prints certain indicia about active processes. To get a complete printout on the console or lpr, use ``ps axlgw'' For a quick snapshot of system activity, ``ps au'' is recommended. A minus may precede options with no effect. The following options may be specified.

a asks for information about all processes with terminals (ordinarily only one's own processes are displayed).

c causes only the comm field to be displayed instead of the arguments. (The comm field is the tail of the path name of the file the process last exec'ed.) This option speeds up ps somewhat and reduces the amount of output. It is also more reliable since the process can't scribble on top of it.

g Asks for all processes. Without this option, ps only prints ``interesting'' processes. Processes are deemed to be uninteresting if they are process group leaders, or if their arguments begin with a `-'. This normally eliminates shells and getty processes.

k causes the file /usr/sys/core is used in place of /dev/kmem and /dev/mem. This is used for postmortem system debugging.

l asks for a long listing. The short listing contains the user name, process ID, tty, the cumulative execution time of the process and an approximation to the command line.

r asks for "raw output". A non-human readable sequence of structures is output on the standard output. There is one structure for each process, the format is defined by <psout.h>

s Print the size of the kernel stack of each process. This may only be used with the short listing, and is for use by system developers.

tttyname
restricts output to processes whose controlling tty is the specified ttyname (which should be specified as printed by ps, e.g. tty3 for tty3, tconsole for console, tttyd0 for ttyd0, t? for processes with no

tty, etc). This option must be the last one given.

u A user oriented output is produced. This includes the name of the owner of the process, process id, nice value, size, resident set size, tty, cpu time used, and the command.

w tells ps you are on a wide terminal (132 columns). Ps normally assumes you are on an 80 column terminal. This information is used to decide how much of long commands to print. The w option may be repeated, e.g. **ww**, and the entire command, up to 128 characters, will be printed without regard to terminal width.

x asks even about processes with no terminal.

# A process number may be given, (indicated here by #), in which case the output is restricted to that process. This option must also be last.

A second argument tells ps where to look for core if the k option is given, instead of /vmcore. A third argument is the name of a swap file to use instead of the default /dev/drum. If a fourth argument is given, it is taken to be the file containing the system's namelist. Otherwise, "/vmunix" is used.

The output is sorted by tty, then by process ID.

The long listing is columnar and contains

F Flags associated with the process. These are defined by #define lines in /usr/include/sys/proc.h.

S The state of the process. 0: nonexistent; S: sleeping; W: waiting; R: running; I: intermediate; Z: terminated; T: stopped.

UID The user id of the process owner.

PID The process ID of the process; as in certain cults it is possible to kill a process if you know its true name.

PPID The process ID of the parent process.

CPU Processor utilization for scheduling.

PRI The priority of the process; high numbers mean low priority.

NICE Used in priority computation.

ADDR The memory address of the process if resident, otherwise the disk address.

SZ The size in blocks of the memory image of the process.

WCHAN
The event for which the process is waiting or sleeping; if blank, the process is running.

TTY The controlling tty for the process.

TIME The cumulative execution time for the process.

COMMAND
The command and its arguments.

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>. Ps makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

## FILES

| | |
|---|---|
| /vmunix | system namelist |
| /dev/kmem | kernel memory |
| /dev/drum | swap device |
| /vmcore | core file |
| /dev | searched to find swap device and tty names |

## SEE ALSO

kill(1), w(1)

## BUGS

Things can change while ps is running; the picture it gives is only a close approximation to reality.

Processes with large environments, which have all or part of the command in a block other than the top block in memory, are not correctly printed by ps, which only looks at the top block in memory. Thus, users using the TERMCAP environment variable will probably only have their command name shown.

## NAME

pstat - print system facts

## SYNOPSIS

**pstat [ -aixptuf ] [ suboptions ] [ file ]**

## DESCRIPTION

Pstat interprets the contents of certain system tables. If file is given, the tables are sought there, otherwise in /dev/mem. The required namelist is taken from /unix. Options are

**-a** Under -p, describe all process slots rather than just active ones.

**-i** Print the inode table with the these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- L locked
- U update time filsys(5)) must be corrected
- A access time must be corrected
- M file system is mounted here
- W wanted by another process (L flag is on)
- T contains a text file
- C changed time must be corrected

CNT Number of open file table entries for this inode.
DEV Major and minor device number of file system in which this inode resides.
INO I-number within the device.
MODE Mode bits, see chmod(2).
NLK Number of links to this inode.
UID User ID of owner.
SIZ/DEV Number of bytes in an ordinary file, or major and minor device of special file.

**-x** Print the text table with these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- T ptrace(2) in effect
- W text not yet written on swap device
- L loading in progress
- K locked
- w wanted (L flag is on)

DADDR Disk address in swap, measured in multiples of 512 bytes.

CADDR Core address, measured in multiples of 64 bytes.

SIZE Size of text segment, measured in multiples of 64 bytes.

IPTR Core location of corresponding inode.

CNT Number of processes using this text segment.

CCNT Number of processes in core using this text segment.

**-p** Print process table for active processes with these headings:

LOC The core location of this table entry.
S Run state encoded thus:
- Ø no process
- 1 waiting for some event
- 3 runnable
- 4 being created
- 5 being terminated
- 6 stopped under trace

F Miscellaneous state variables, or-ed together:
- Ø1 loaded
- Ø2 the scheduler process
- Ø4 locked
- Ø1Ø swapped out
- Ø2Ø traced
- Ø4Ø used in tracing
- Ø1ØØ locked in by lock(2).

PRI Scheduling priority, see nice(2).
SIGNAL
Signals received (signals 1-16 coded in bits Ø-15),
UID Real user ID.
TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see nice(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
ADDR If in core, the physical address of the `u-area' of the process measured in multiples of 64 bytes. If swapped out, the position in the swap area measured in multiples of 512 bytes.
SIZE Size of process image in multiples of 64 bytes.
WCHAN Wait channel number of a waiting process.
LINK Link pointer in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for alarm(2) measured in seconds.

**-t** Print table for terminals (only DH11 and DL11 handled) with these headings:

RAW Number of characters in raw input queue.
CAN Number of characters in canonicalized input queue.
OUT Number of characters in putput queue.
MODE See tty(4).
ADDR Physical device address.
DEL Number of delimiters (newlines) in canonicalized input queue.
COL Calculated column position of terminal.
STATE Miscellaneous state variables encoded thus:
W waiting for open to complete
O open
S has special (output) start routine
C carrier is on
B busy doing output
A process is awaiting output
X open for exclusive use
H hangup on close
PGRP Process group for which this is controlling terminal.

**-u** print information about a user process; the next argument is its address as given by ps(1). The process must be in main memory, or the file used can be a core image and the address Ø.

**-f** Print the open file table with these headings:

LOC The core location of this table entry.
FLG Miscellaneous state variables encoded thus:
R open for reading
W open for writing
P pipe
CNT Number of processes that know this open file.
INO The location of the inode table entry for this file.
OFFS The file offset, see lseek(2).

**FILES**

/unix namelist
/dev/mem default source of tables

**SEE ALSO**

ps(1), stat(2), filsys(5)
K. Thompson, UNIX Implementation

**NAME**

ptx - permuted index

**SYNOPSIS**

ptx [ option ] ... [ 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 the page. Ptx produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an nroff or troff(1) macro for user-defined formatting. The before keyword and keyword and after fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. Tail and head, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

-f Fold upper and lower case letters for sorting.

-t Prepare the output for the phototypesetter; the default line length is 100 characters.

-w n Use the next argument, n, as the width of the output line. The default line length is 72 characters.

-g n Use the next argument, n, as the number of characters to allow 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. In any case, tab, newline, and space characters are always used as break characters.

-r Take any leading nonblank 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.

**NAME**

pwd - working directory name

**SYNOPSIS**

pwd

**DESCRIPTION**

Pwd prints the pathname of the working (current) directory.

**SEE ALSO**

cd(1)

## NAME

ranlib - convert archives to random libraries

## SYNOPSIS

ranlib archive ...

## DESCRIPTION

Ranlib converts each archive to a form which can be loaded more rapidly by the loader, by adding a table of contents named __.SYMDEF to the beginning of the archive. It uses ar(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system containing the current directory.

## SEE ALSO

ld(1), ar(1)

## BUGS

Because generation of a library by ar and randomization by ranlib are separate, phase errors are possible. The loader ld warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

**NAME**

ratfor - rational Fortran dialect

**SYNOPSIS**

**ratfor** [ option ... ] [ filename ... ]

**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 [n]
     next [n]
```

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)
     returns expression to caller from function

define:
     define name replacement

include:
     include filename
```

The option -h causes quoted strings to be turned into 27H constructs. -C 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).

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, Software Tools, Addison-Wesley, 1976.

**NAME**

restor - incremental file system restore

**SYNOPSIS**

restor key [ argument ... ]

**DESCRIPTION**

Restor is used to read magtapes 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 icheck -s must be done before

x Each file on the tape named by an argument is extracted. The file name has all `mount' prefixes removed; for example, /usr/bin/lpr is named /bin/lpr on the tape. The file extracted 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:

Mount volume 1 of the set of dump tapes.

Type the restor command.

Restor will announce whether or not it found the files, give the number it will name the file, and rewind the tape.

It then asks you to `mount the desired tape volume'. Type the number of the volume you choose. On a multivolume dump the recommended procedure is to mount the last through the first volume in that order. Restor checks to see if any of the files requested 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 the number of files being restored is large, respond to the query with `1' and restor will read the tapes in sequential order.

If you have a hierarchy to restore you can use dumpdir(1) to produce the list of names and a shell

script to move the resulting files to their homes.

t Print the date the tape was written and the date the filesystem 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 this.

A dump followed by a mkfs and a restor is used to change the size of a file system.

## FILES

default tape unit varies with installation
rst*

## SEE ALSO

dump(1), mkfs(1), dumpdir(1)

## 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

rm, rmdir - remove (unlink) files

## 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 (force) option is given.

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 .*'.

## 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 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 specified must not appear in any entry of 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

rmdir, rm - remove (unlink) directories or files

## SYNOPSIS

**rmdir dir ...**

**rm [ -f ] [ -r ] [ -i ] [ - ] file ...**

## DESCRIPTION

Rmdir removes entries for the named directories, which must be empty.

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 and no errors are reported when the -f (force) option is given.

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.

The null option - indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

## SEE ALSO

rm(1), unlink(2), rmdir(2)

## MESSAGES

rmdir: arg count
: You must specify at least one directory name for rmdir.

rmdir: cannot remove . or ..
: You cannot remove the current directory or the parent directory of the current directory.

rmdir: directory non-existent
: The directory specified by directory could not be

found - check the spelling.

rmdir: cannot stat
The indicated directory cannot be found, or you don't have search permission for it.

rmdir: directory not a directory
The object specified by directory is not a directory - rmdir will not remove ordinary files.

rmdir: cannot remove current directory
You cannot remove the current directory.

rmdir: directory unreadable
The directory specified by directory cannot be read - the permissions are probably not correct.

rmdir: directory not empty
A directory cannot be removed while it still has files in it.

rmdir: directory: no permission
The parent directory containing the directory specified by directory does not have write permission.

rmdir: directory not removed
This message indicates you are trying to remove a directory you don't own.

## 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

sccsdiff - compare two versions of an SCCS file

## SYNOPSIS

**sccsdiff -rSID1 -rSID2 [-p] [-sn] 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 Guideby L. E. Bonanni and C. A. Salemi.

## DIAGNOSTICS

``file: No differences'' If the two versions are the same. Use help(1) for explanations.

**NAME**

sed - stream editor

**SYNOPSIS**

**sed [ -n ] [ -e script ] [ -f sfile ] [ file ] ...**

**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's, 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.

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, `/regular expression/', in the style of ed(1) modified thus:

The escape sequence `\n' matches a newline embedded in 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.

An argument denoted text consists of one or more lines, all but the last of which end with `\' to hide the newline. 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.

An argument denoted rfile or wfile 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 newline. 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 newline. (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 newline 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).

(Ø): 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.

(Ø) An empty command is ignored.

SEE ALSO
ed(1), grep(1), awk(1)

## NAME

sfnt - select loaded font

## SYNOPSIS

**sfnt** [ fontnum ]

## DESCRIPTION

Sfnt selects the current font in use to be fontnum. The font must have already been loaded into the window using **lfnt**. Fontnum must be in the range Ø to 7. If fontnum is not specified, it defaults to 7 which is the default font for the window.

## SEE ALSO

lfnt(1) lsfnt(1) cfnt(1)

**NAME**

sfsck - file system consistency check and interactive repair

**SEE**

fsck(1)

## NAME

sh, for, case, if, while, :, ., break, continue, cd, eval, exec, exit, export, login, newgrp, read, readonly, set, shift, times, trap, umask, wait - command language

## SYNOPSIS

**sh** [ **-ceiknrstuvx** ] [ arg ] ...

## DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See **invocation** 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 Ø (see exec(2)). The value of a simple-command is its exit status if it terminates normally or 2ØØ+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 **&**. **;** and **&** have equal precedence which is lower than that of **&&** and **||**, **&&** and **||** also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding pipeline to be executed without waiting for it to finish. The symbol **&&** (**||**) causes the list following to be executed only if the preceding pipeline returns a zero (non zero) value. Newlines may appear in a list, instead of semicolons, to delimit commands.

A command is either a simple-command or one of the following. 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 in the **for** word list If **in** word ... is omitted then **in "$@"** is assumed. 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.

**if** list **then** list [**elif** list **then** list] ... [**else** list] **fi**
The list following **if** is executed and if it returns zero the list following **then** is executed. Otherwise, the list following **elif** is executed and if its value is zero the list following **then** is executed. Failing that the **else** list is executed.

**while** list [**do** list] **done**
A **while** command repeatedly executes the **while** list and if its value is zero executes the **do** list; otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** list. **until** may be used in place of **while** to negate the loop termination test.

**(** list **)**
Execute list in a subshell.

**{** 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 in esac for while until do done { }**

**Command substitution.**
The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing newlines are removed.

**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 ] ...

${parameter}
A parameter is a sequence of letters, digits or underscores (a name), a digit, or any of the characters *** @ # ? - $ !**. 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. 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. $Ø is set from argument zero when the shell is invoked.

${parameter-word}
If parameter is set then substitute its value; otherwise substitute word.

${parameter=word}
If parameter is not set 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 then substitute its value; otherwise, print word and exit from the shell. If word is omitted then a standard message is printed.

${parameter+word}
If parameter is set 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, for example, echo ${d-`pwd`} will only execute pwd if d is unset.)

The following parameters are automatically set by the shell.

| | |
|---|---|
| # | The number of positional parameters in decimal. |
| - | Options supplied to the shell on invocation or by set. |
| ? | The value returned by the last executed command in decimal. |
| $ | The process number of this shell. |
| ! | The process number of the last background command invoked. |

The following parameters are used but not set by the shell.

| | |
|---|---|
| HOME | The default argument (home directory) for the cd command. |
| PATH | The search path for commands (see execution). |
| 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 newline. |

Blank interpretation.

After parameter and command substitution, any 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 /, and the character /, must be matched explicitly.

***** Matches any string, including the null string.
**?** Matches any single character.
**[...]**
Matches any one of the characters enclosed. A pair of characters separated by - matches any character lexically between the pair.

**Quoting.**
The following characters have a special meaning to the shell and cause termination of a word unless quoted.

**; & ( ) | < > newline space tab**

A character may be quoted by preceding it with a \. **\newline** is ignored. All characters enclosed between a pair of quote marks (''), except a single quote, are quoted. Inside double quotes ("") parameter and command substitution occurs and \ quotes the characters \ ` " and $.

**"$*"** is equivalent to **"$1 $2 ..."** whereas
**"$@"** is equivalent to **"$1" "$2" ...** .

**Prompting.**
When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a newline is typed and further input is needed to complete a command then the secondary prompt ($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 Ø).

>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 (by seeking to the end); otherwise the file is created.

<<word
The shell input is read up to a line the same as word, or 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, \newline is ignored, and \ is used to quote the characters \ $ ` and the first character of word.

<&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 Ø or 1). For example,

... 2>&1

creates file descriptor 2 to be 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 is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see exec(2) and environ(5). 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 these two lines are equivalent

```
TERM=45Ø cmd args
(export TERM; TERM=45Ø; cmd args)
```

If the **-k** flag is set, all keyword arguments are placed in the environment, even if the occur after the command name. The following prints `a=b c' and `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. (But see also **trap**.)

**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 an exec(2).

**The shell parameter $PATH defines the search path for the directory containing the command. Each alternative** directory name is separated by a colon (:). The default path is **:/bin:/usr/bin**. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an a.out file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

**Special commands.**
The following commands are executed in the shell process and except where specified no input output redirection is

permitted for such commands.

**:** No effect; the command does nothing.

**. file**
Read and execute commands from file and return. The search path $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 non interactive 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 exit from the shell.)

**export [name ...]**
The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given then a list of exportable names is printed.

**login [arg ...]**
Equivalent to `exec login arg ...'.

**newgrp [arg ...]**
Equivalent to `exec newgrp arg ...'.

**read name ...**
One line is read from the standard input; successive words of the input are assigned to the variables name in order, with leftover words to the last variable. The return code is Ø unless the end-of-file is encountered.

**readonly [name ...]**
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given then a list of all readonly names is printed.

**set [-eknptuvx [arg ...]]**

**-e** If non interactive then exit immediately if a command fails.
**-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.
**-** Turn off the **-x** and **-v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in $-.

Remaining arguments are positional parameters and are assigned, in order, to $1, $2, etc. If no arguments are given then the values of all names are printed.

**shift**
The positional parameters from $2... are renamed $1...

**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. 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 invoked commands. If n is Ø then the command arg is executed on exit from the shell, otherwise upon receipt of signal n as numbered in signal(2). Trap with no arguments prints a list of commands associated with each signal number.

**umask** [ nnn ]
The user file creation mask is set to the octal value nnn (see umask(2)). If nnn is omitted, the current value of the mask is printed.

**wait** [n]
Wait for the specified process and report its termination status. If n is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

**Invocation.**
If the first character of argument zero is -, commands are read from **$HOME/.profile**, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

**-c** string — If the **-c** flag is present then commands are read from string.

**-s** — If the **-s** flag is present or if no arguments remain then commands are read from the standard input. Shell output is written to file descriptor 2.

**-i** — If the **-i** flag is present or if the shell input and output are attached to a terminal (as told by gtty) then this shell is interactive. In this case the terminate signal SIGTERM (see signal(2)) is ignored (so that `kill Ø' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptable). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command.

**FILES**

$HOME/.profile
/tmp/sh*
/dev/null

**SEE ALSO**

test(1), exec(2),

DIAGNOSTICS

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 **exit**).

**BUGS**

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 the file by another name.

**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(3)

**BUGS**

Time must be less than 65536 seconds.

## NAME

sort - sort or merge files

## SYNOPSIS

**sort** [ -mubdfinrtx ] [ +pos1 [ -pos2 ] ] ... [ -o name ] [ -T directory ] [ name ] ...

## 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 Ø4Ø-Ø176 in nonnumeric 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 nonblank in the field; **b** is attached independently to pos2. A missing .n means .Ø; a missing -pos2 means the end of the line. Under the -tx option, fields are strings separated by x; otherwise fields are nonempty nonblank 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.

**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.

**T** The next argument is the name of a directory in which temporary files should be made.

**u** Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

**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 number (the 3rd 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*, /tmp/*: first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option -c.

## BUGS

Very long lines are silently truncated.

## NAME

spell, spellin, spellout - find spelling errors

## SYNOPSIS

**spell** [ option ] ... [ file ] ...

**/src/cmd/spell/spellin** [ list ]

## DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes 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, tbl and eqn(1) constructions.

Under the **-v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words 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.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in 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.

Spellin augments the hash list used by spell. It expects 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.

## FILES

D=/usr/dict/hlist[ab]: hashed spelling lists, American & British
S=/usr/dict/hstop: hashed stop list
H=/usr/dict/spellhist: history file
/usr/lib/spell

deroff(1), sort(1), tee(1), sed(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.
British spelling was done by an American.

## NAME

spline - interpolate smooth curve

## SYNOPSIS

**spline** [ option ] ...

## DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas 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., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by graph(1).

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

(2nd deriv. at end) = k*(2nd deriv. next to end)

is set by the next argument. By default k = Ø.

**-n** Space output points so that approximately n intervals occur between the lower and upper x limits. (Default n = 1ØØ.)

**-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 abcissas start at lower limit (default Ø).

## SEE ALSO

graph(1)

## DIAGNOSTICS

When data is not strictly monotone in x, spline reproduces the input without interpolating extra points.

## BUGS

A limit of 1ØØØ 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.

## 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 loader. 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.

## FILES

/tmp/stm? temporary file

## SEE ALSO

ld(1)

## NAME

stty - set terminal options

## SYNOPSIS

stty [ option ... ]

## DESCRIPTION

Stty sets certain I/O options on the current output terminal. With no argument, it reports the current settings of the options. The option strings are selected from the following set:

| | |
|---|---|
| even | allow even parity |
| -even | disallow even parity |
| odd | allow odd parity |
| -odd | disallow odd parity |
| raw | raw mode input (no erase, kill, interrupt, quit, EOT; parity bit passed back) |
| -raw | negate raw mode |
| cooked | same as `-raw' |
| cbreak | make each character available to read(2) as received; no erase and kill |
| -cbreak | make characters available to read only when newline is received |
| -nl | allow carriage return for new-line, and output CR-LF for carriage return or new-line |
| nl | accept only new-line to end lines |
| echo | echo back every character typed |
| -echo | do not echo characters |
| lcase | map upper case to lower case |
| -lcase | do not map case |
| -tabs | replace tabs by spaces when printing |
| tabs | preserve tabs |
| ek | reset erase and kill characters back to normal # and @ |
| erase c | set erase character to c (default `#'.) |
| kill c | set kill character to c (default `@'.) |
| intr c | set interrupt character to c (default DEL.) |
| quit c | set quit character to c (default control \.) |
| start c | set start character to c (default control Q.) |
| stop c | set stop character to c (default control S.) |
| eof c | set end of file character to c (default control D.) |
| brk c | set break character to c (default undefined.) This character is an extra wakeup causing character. |
| crØ cr1 cr2 cr3 | select style of delay for carriage return (see ioctl(2)) |
| nlØ nl1 nl2 nl3 | select style of delay for linefeed |
| tabØ tab1 tab2 tab3 | select style of delay for tab |
| ffØ ff1 | select style of delay for form feed |

**bs0 bs1** select style of delay for backspace
**tty33** set all modes suitable for the Teletype Corporation Model 33 terminal.
**tty37** set all modes suitable for the Teletype Corporation Model 37 terminal.
**vt05** set all modes suitable for Digital Equipment Corp. VT05 terminal
**tn300** set all modes suitable for a General Electric TermiNet 300
**ti700** set all modes suitable for Texas Instruments 700 series terminal
**tek** set all modes suitable for Tektronix 4014 terminal
**hup** hang up dataphone on last close.
**-hup** do not hang up dataphone on last close.
**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. (These are the speeds supported by the DH-11 interface).

**SEE ALSO**
ioctl(2), tabs(1), tset(1), stty(1)

**NAME**

su - substitute user id temporarily

**SYNOPSIS**

su [ userid ]

**DESCRIPTION**

Su demands the password of the specified userid, and if it is given, changes to that userid and invokes the Shell sh(1) without changing the current directory or the user environment (see environ(5)). The new user ID stays in force until the Shell exits.

If no userid is specified, `root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes `#' for its usual prompt.

**SEE ALSO**

sh(1)

## NAME

sum - sum and count blocks in a file

## SYNOPSIS

sum 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.

## SEE ALSO

wc(1)

## DIAGNOSTICS

`Read error' is indistinuishable 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. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See sync(2) for details.

**SEE ALSO**

sync(2).

**NAME**

tail - deliver the last part of a file

**SYNOPSIS**

**tail** +number[lbc] [ 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. 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.

**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**

tar - tape archiver

**SYNOPSIS**

**tar** [ key ] [ name ... ]

**DESCRIPTION**

Tar saves and restores files on magtape. 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 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.

**x** The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

**t** The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

**u** The named files are added to the tape if either they are not already there or have been modified since last put on the tape.

**c** Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies **r**.

The following characters may be used in addition to the letter which selects the function desired.

**Ø,...,7** This modifier selects the drive on which the tape is mounted. The default is 1.

**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 file name, then wait for user confirmation. If a word beginning with `y' is given, the action is performed. Any other input means don't do it.

f causes tar to use the next argument as the name of the archive instead of /dev/mt?. If the name of the file is `-', tar writes to standard output or reads from standard input, whichever is appropriate. Thus, tar can be used as the head or tail of a filter chain 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 2Ø. This option should only be used with raw magnetic tape archives (See **f** above).

l tells tar to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error meesages are printed.

p when used with x (extract), tells tar to preserve the ownership and mode bits of each file.

**FILES**

/dev/mt?
/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 magtape driver cannot backspace raw magtape. If the archive is on a disk file the b option should not be used at all, as updating an archive stored in this manner can destroy it.

The current limit on file name length is 100 characters.

## NAME

tbl - format tables for nroff or troff

## SYNOPSIS

tbl [ files ] ...

## DESCRIPTION

Tbl is a preprocessor for formatting tables for nroff 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 reformatted. Details are given in the reference manual.

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.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 |
| Branchburg | 1644 | 3.49 |
| Bridgewater | 7897 | 3.81 |
| Far Hills | 240 | 3.19 |

If no arguments are given, tbl reads the standard input, so it may be used as a filter. When it is used with eqn or neqn the tbl command should be first, to minimize the volume of data passed through pipes.

**SEE ALSO**

troff(1), eqn(1)
M. E. Lesk, TBL.

**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. Option -i ignores interrupts; option -a causes the output to be appended to the files rather than overwriting them.

## NAME

test - condition command

## SYNOPSIS

**test** expr

## DESCRIPTION

test evaluates the expression expr, and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. test returns a non zero exit if there are no arguments.

The following primitives are used to construct expr.

**-r** file true if the file exists and is readable.

**-w** file true if the file exists and is writable.

**-f** file true if the file exists and is not a directory.

**-d** file true if the file exists and is a directory.

**-s** file true if the 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 nonzero.

s1 **=** s2 true if the strings s1 and s2 are equal.

s1 **!=** s2 true if the strings s1 and s2 are not equal.

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**, or **-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

( expr )
parentheses for grouping.

-a has higher precedence than -o. Notice that all the operators and flags are separate arguments to test. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO
sh(1), find(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 the diagnostic output stream.

## BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

**NAME**

tip - connect to a remote system

**SYNOPSIS**

**tip [ -v ] [ -speed ] system-name**

**DESCRIPTION**

Tip establishes a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote cpu. It goes without saying that you must have a login on the machine (or equivalent) to which you wish to connect.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde (``~'') appearing as the first character of a line is an escape signal; the following are recognized:

~^D or ~. Drop the connection and exit (you may still be logged in on the remote machine).

~c [name] Change directory to name (no argument implies change to your home directory).

~! Escape to a shell (exiting the shell will return you to tip).

~> Copy file from local to remote.

~< Copy file from remote to local.

~p from [to]
Send a file to a remote UNIX host. The put command causes the remote UNIX system to run the command string ``cat > 'to''', while tip sends it the ``from'' file. If the ``to'' file isn't specified the ``from'' file name is used. This command is actually a UNIX specific version of the ``~>'' command.

~t from [to]
Take a file from a remote UNIX host. As in the put command the ``to'' file defaults to the ``from'' file name if it isn't specified. The remote host executes the command string ``cat 'from';echo ^A'' to send the file to tip.

~| Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.

~# Send a BREAK to the remote system. For systems

which don't support the necessary ioctl call the break is simulated by a sequence of line speed changes and DEL characters.

~s Set a variable (see the discussion below).

~^Z Stop tip (only available with job control).

~? Get a summary of the tilde escapes

Copying files requires some cooperation on the part of the remote host. When a ~> or ~< escape is used to send a file, tip will prompt for a file name (to be transmitted or received) and a command which will be sent to the remote system, in case the file is being transferred from the remote system. The default end of transmission string for transferring a file from the local system to the remote is specified in the remote(5) file, but may be changed by the set command. While tip is transferring a file the number of lines transferred will be continuously displayed on the screen. A file transfer may be aborted with an interrupt. An example of the dialogue used to transfer files is given below.

```
% tip mds
[connected]
<<Assume we are talking to another UNIX system>>
login: ken
Password:
% cat > foo.c
~> Filename: foo.c
32 lines transferred in 1 minute 3 seconds
%
% ~< Filename: reply.c
List command for remote host: cat reply.c
65 lines transferred in 2 minutes
%
<<Or, equivalently>>
% ~p foo.c
<<actually echo's as ~[put] foo.c>>
32 lines transferred in 1 minute 3 seconds
%
% ~t reply.c
<<actually echo's as ~[take] reply.c>>
65 lines transferred in 2 minutes
%
<<To print a remote file locally>>
% ~|Local command: pr -h foo.c | lpr
List command for remote host: cat foo.c
% ~^D
[EOT]
```

```
<<Back on the local system>>
%
```

Tip can be used to connect to the sdu of a Nu Machine. If the sdu operating system is running (and not UNIX), then the only escape sequences that can be used are the local ones and sending a break to the sdu. The ttf driver in the sdu operating system sends escape sequences to tip to access files on the local system.

The remote(5) file contains the definitions for all remote systems known by tip; refer to the remote manual page for a full description. In particular, each system-name has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, e.g. ``tip -300 mds''.

When tip establishes a connection it sends out a connection message to the remote system. The default value for this string may be found in the remote file.

At any time that tip prompts for an argument (e.g. during set-up of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote machine.

When tip attempts to connect to a remote system, it opens the associated device with an exclusive-open ioctl(2) call. Thus only one user at a time may access a device. This is to prevent multiple processes from sampling the terminal line. In addition, tip honors the locking protocol used by uucp(1).

## AUTO-CALL UNITS

Tip may be used to dial up remote systems using a number of auto-call unit's (ACU's). When the remote system description contains the ``du'' attribute, tip will use the call-unit (``cu''), ACU type (``at''), and phone numbers (``pn'') supplied. Normally tip will print out the verbose messages as it dials. See remote(5) for details of the remote host specification.

Depending on the type of auto-dialer being used to establish a connection the remote host may have garbage characters sent to it upon connection. The user should never assume that the first characters typed to the foreign host are the first ones presented to it. The recommended practice is to immediately type a ``kill'' character upon establishing a connection (most UNIX systems support ``@'' as the initial kill character).

Tip currently supports DEC's DN-11 interface with Bell 801 or Vadic 821 callers, DEC's DF02-AC and DF03-AC, Bizcomp's 1031 and 1022 intelligent modems, Ventel's autodialer modems, and Vadic VA3450 modem.

## REMOTE HOST DESCRIPTIONS

Descriptions of remote hosts are normally located in the system-wide file /etc/remote. However, a user may maintain personal description files (and phone numbers) by defining and exporting the shell variable **REMOTE**. The remote file must be readable by tip, but a secondary file describing phone numbers may be maintained readable only by the user. This secondary phone number file defaults to /etc/phones, unless the shell variable **PHONES** is defined and exported. As described in remote(5), the phones file is read when the host description's phone number(s) capability is an ``@''. The phone number file contains lines of the form:

```
system-name     phone-number
```

Each phone number found for a system is tried until either a connection is established, or an end of file is reached. Phone numbers are constructed from "0123456789-=*", where the ``='' and ``*'' are used to indicate a second dial tone should be waited for (ACU dependent).

## VARIABLES

Tip maintains a set of variables which are used in normal operation. Some of these variable are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the ``s'' escape. The syntax for variables is patterned after the vi(1) editor and the mail(1) system. Supplying ``all'' as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a ``?'' to the end. For example ``escape?'' would display the current escape character.

Variables are numeric, string, character, or boolean values. Boolean variables are set merely by specifying their name. They may be reset by prepending a ``!'' to the name. Other variable types are set by appending an ``='' and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run time by placing set commands (without the ```s'' prefix in a file .tiprc in one's home directory). The -v option causes tip to display the sets as they are made.

Finally, the variable names must either be completely specified or an abbreviation may be given. The following list details those variables known to tip, their

abbreviations (surrounded by brackets), and their default values. Those variables initialized from the remote file are marked with a ``*''. A mode is given for each variable; capitalization indicates the read or write capability is given only to the super-user.

| Name | Type | Mode | Default | Description |
|---|---|---|---|---|
| [be]autify | bool | rw | true | discard unprintables when scripting |
| [ba]udrate | num | rW | * | connection baud rate |
| [dial]timeout | num | rW | 60 | timeout (seconds) when establishing connection |
| [eofr]ead | str | rw | * | char's signifying EOT from the remote host |
| [eofw]rite | str | rw | * | string sent for EOT |
| [eol] | str | rw | * | char's signifying an end of lin |
| [es]cape | char | rw | `~' | command prefix character |
| [ex]ceptions | str | rw | "\t\n\f\b" | char's not discarded due to beautification |
| [fo]rce | char | rw | `^p' | forces transparency on next cha |
| [fr]amesize | num | rw | * | size of buffering between writes on reception |
| [ho]st | str | r | * | name of host connected to |
| [lock] | str | RW | s.d. | lock file for ACU logging |
| [log] | str | RW | s.d. | ACU log file |
| [phones] | str | r | s.d. | file for hidden phone numbers |
| [pr]ompt | char | rW | `\n' | end of line indicator set by ho |
| [ra]ise | bool | rw | false | upper case mapping switch |
| [r]aise[c]har | char | rw | `^A' | interactive toggle for raise |
| [rec]ord | str | rw | "tip.record" | name of script output file |
| [remote] | str | r | s.d. | system description file |
| [sc]ript | bool | rw | false | session scripting switch |
| [tab]expand | bool | rw | false | expand tabs during file transfe |
| [verb]ose | bool | rw | true | make noise during file transfer |
| [SHELL] | str | rw | "/bin/sh" | name of shell for ~! escape |
| [HOME] | str | rw | "" | home directory for ~c escape |

NOTE: 1. s.d. indicates the default value is system dependent; normally these values are lock=/tmp/aculock, log=/usr/adm/aculog, phones=/etc/phones, and remote=/etc/remote.

ENVIRONMENT VARIABLES

The following variables are read from the environment:

REMOTE The location of the remote file.

PHONES The location of the file containing private phone numbers.

HOST A default host to connect to.

HOME One's log-in directory (for chdirs).

SHELL The shell to fork on a ~! escape.

**FILES**

| | |
|---|---|
| ~/.tiprc | Variable initialization file. |
| /usr/spool/uucp/LCK..* | Lock file to avoid conflicts with uucp. Terminated with port name. |

**DIAGNOSTICS**

unrecognizable host name
The phone number file is malformed.

missing phone number
A line in the phone number file is incomplete; or the remote host description indicates the host should be called, but no phone number is specified.

bizcomp out of sync
When using a BIZCOMP ACU the dialer must be ``synced'' up before tip will attempt to place a call. Try running tip again.

missing device spec
The remote host description entry is missing a device specification.

timedout at eol n
During a file transfer tip has lost contact with the remote system while waiting for a synchronizing echo.

timeout error
Same as above.

Other diagnostics are, hopefully, self explanatory.

**SEE ALSO**

remote(5)

**NAME**

touch - update date last modified of a file

**SYNOPSIS**

**touch [ -c ] file ...**

**DESCRIPTION**

Touch attempts to set the modified date of each file. This is done by reading a character from the file and writing it back.

If a file does not exist, an attempt will be made to create it unless the -c option is specified.

## 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. When string2 is short it is padded to the length of string1 by duplicating its last character. 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 Ø1 through Ø377 octal; -d deletes all input characters in string1; -s squeezes all strings of repeated output characters that are in string2 to single characters.

In either string the notation a-b means a range of characters from a to b in increasing ASCII order. The character `\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A `\' followed by any other character stands for that character.

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 second string is quoted to protect `\' from the Shell. Ø12 is the ASCII code for newline.

```
tr -cs A-Za-z '\Ø12' <file1 >file2
```

## SEE ALSO

ed(1), ascii(7)

## BUGS

Won't handle ASCII NUL in string1 or string2; always deletes NUL from input.

## 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

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 terminal name

**SYNOPSIS**

**tty**

**DESCRIPTION**

Tty prints the pathname of the user's terminal.

**DIAGNOSTICS**

`not a tty' if the standard input file is not a terminal.

**NAME**

ul - do underlining

**SYNOPSIS**

**ul** [ **-t** terminal ] [ name ... ]

**DESCRIPTION**

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining. If **-t** is present, terminal is used as the terminal kind. Otherwise, the environment is looked in and /etc/termcap read to determine the appropriate sequences for underlining. If none of the fields **us**, **ue**, or **uc** is present, and if **so** and **se** are present, standout mode is used to indicate underlining. If the terminal can overstrike, or handles underlining automatically, ul behaves like cat(1). If the terminal cannot underline, underlining is ignored.

**SEE ALSO**

iul(1), man(1), nroff(1)

**AUTHOR**

Mark Horton

**BUGS**

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

**NAME**

umount - file system dismount

**SEE**

mount(1)

**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.

-rSID — 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 gets 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. 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

sort(1), comm(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
     * 2.54000e+00
     / 3.93701e-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 pounds force/in2
You want: atm
     * 1.02069e+00
     / 9.79730e-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 a unit of mass. Compound names are run together, e.g. `lightyear'. British units that differ from their US counterparts are prefixed thus: `brgallon'. Currency is denoted `belgiumfranc', `britainpound', ...

For a complete list of units, `cat /usr/lib/units'.

## FILES

/usr/lib/units

## BUGS

Don't base your financial plans on the currency conversions.

## NAME

uucp, uulog - unix to unix copy

## SYNOPSIS

**uucp** [ option ] ... source-file ... destination-file

**uulog** [ option ] ...

## 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!pathname

where `system-name' is taken from a list of system names which uucp knows about. Shell metacharacters ?*[] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~user; where user is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

If the result is an erroneous pathname 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 Ø666 read and write permissions (see chmod(2)).

The following options are interpreted by uucp.

**-d** Make all necessary directories for the file copy.

**-c** Use the source file when copying out rather than copying the file to the spool directory.

**-m** Send mail to the requester when the copy is complete.

Uulog maintains a summary log of uucp and uux(1) transactions in the file `/usr/spool/uucp/LOGFILE' by gathering information from partial log files named `/usr/spool/uucp/LOG.*.?'. It removes the partial log files.

The options cause uulog to print logging information:

-ssys
Print information about work involving system sys.

-uuser
Print information about work done for the specified user.

**FILES**
/usr/spool/uucp - spool directory
/usr/lib/uucp/* - other data and program files

**SEE ALSO**
uux(1), mail(1)
D. A. Nowitz, Uucp Implementation Description

**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 pathname; 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 pathnames.

**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**

uulog - unix to unix copy

**SEE**

uucp(1)

**NAME**

uux - unix to unix command execution

**SYNOPSIS**

**uux** [ - ] command-string

**DESCRIPTION**

Uux will gather Ø or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

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 pathname;

(2) a pathname preceded by ~xxx; where xxx is a userid 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 > !fi.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.

**FILES**

/usr/uucp/spool - spool directory
/usr/uucp/* - other data and programs

**SEE ALSO**

uucp(1)
D. A. Nowitz, Uucp implementation description

**WARNING**

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an

incoming request from uux. Typically, a restricted site will permit little other than the receipt of mail via uux.

**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.
There is no notification of denial of execution on the remote machine.

**NAME**

val - validate SCCS file

**SYNOPSIS**

**val -**
**val** [-s] [-rSID] [-mname] [-ytype] 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 (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the SID is ambiguous (e. g., rl is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., rl.Ø or rl.1.Ø 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 %M% keyword in file.

**-y**type The argument value type is compared with the SCCS %Y% 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 Ø = 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%, -y mismatch;
bit 7 = %M%, -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 5Ø files on a single command line. Any number above 5Ø will produce a core dump.

**NAME**

vc - version control

**SYNOPSIS**

vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

**DESCRIPTION**

The vc command copies lines from the standard input to the standard output under control of its arguments and control statements encountered in the standard input. In the process of performing the copy operation, user declared keywords may be replaced by their string value when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, except as modified by the -t keyletter (see below). The default control character is colon (:), except as modified by the -c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or 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.: Ø12 > 12 is false). All other operators take strings as arguments (e. g.: Ø12 != 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
Ø - normal
1 - any error

**NAME**

vi - screen oriented (visual) display editor based on ex

**SYNOPSIS**

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).

**SEE ALSO**

ex(1), vi(1), ``Vi Quick Reference'' card, ``An Introduction to Display Editing with Vi''.

**BUGS**

Scans with / and ? begin on the next line, skipping the remainder of the current line.

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**

view - screen oriented (visual) display editor based on ex

**SEE**

vi(1)

## 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, preceded by `Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

## FILES

/dev/tty?
/etc/utmp

## 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** [ **-lwcpt** ] [ **-b**_baud_ ] [ **-s**_pagesize_ ] [ **-u** ] [ **-v** ] [ name ... ]

## DESCRIPTION

_Wc_ counts lines, words and characters, and optionally pages and the print time, in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If an argument beginning with one of ``lwcpt'' is present, the specified counts (lines, words, characters, pages, or time) are selected by the letters **l**, **w**, **c**, **p**, or **t**. The default is **-lwc** unless **-v** is specified.

The **-b** option asks that the time be figured at the specified baud rate instead of the default 300 baud.

The **-s** option specifies that pages are _pagesize_ lines long instead of the default 66.

The **-u** options asks that the time printed be based on uucp transmission time, about 90% as fast as normal.

The **-v** option asks for a verbose output format, with headers and including pages and time by default.

## BUGS

The times given do not take into account variable factors such as system load; delays due to tab expansion or tty driver delays, which can be a factor with cu; or uucp delays such as mail headers, auxillary protocol files, or the time taken to initially connect to another site.

## 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 %Z% (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 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**

whoami - print effective current user id

**SYNOPSIS**

**whoami**

**DESCRIPTION**

Whoami prints who you are. It works even if you are su'd, while `who am i' does not since it uses /etc/utmp.

**FILES**

/etc/passwd Name data base

**SEE ALSO**

who (1)

**AUTHOR**

Bill Joy

## NAME

write - write to another user

## SYNOPSIS

write user [ ttyname ]

## DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message

Message from yourname yourttyname...

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 `EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the ttyname argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the mesg command. At the outset writing is allowed. Certain commands, in particular nroff 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 to write back before starting to send. Each party should end each message with a distinctive signal-(o) for `over' is conventional-that the other may reply. (oo) for `over and out' is suggested when conversation is about to be terminated.

## FILES

/etc/utmp to find user
/bin/sh to execute `!'

## SEE ALSO

mesg(1), who(1), mail(1)

## NAME

wtty - set window modes

## SYNOPSIS

**wtty [ [-]mode ] ... [ spec [ val ] ] ... [ all ]**

## DESCRIPTION

Wtty sets or displays window modes, size, and position parameters. The following modes can be enables. If preceded by a minus sign, the the mode is disabled.

**scroll** Causes the text in the window to scroll up when at bottom of the window.

**wrap** Causes the line to wrap around when end of line is reached.

**csr** Enables the cursor.

**label** The label at the top of the window appears.

**save** Textual data in an obscured window is saved and then restored when the window is no longer obscured.

**bflsp** Sets big font line spacing.

Spec parameters allow setting window size and position. The val parameter specifies either a coordinate or dimension in pixels.

**uclx** X-coordinate of upper left-hand corner of the window.

**ulcy** Y-coordinate of upper left-hand corner of the window.

**width** Specifies the width of the window.

**height** Specifies the height of the window.

**all** Displays the current modes, postions and size of the window.

## NAME

xstr - extract strings from C programs to implement shared strings

## SYNOPSIS

**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 appropriate 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 |

**SEE ALSO**

mkstr(1)

**AUTHOR**

Bill Joy

**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.

# Nu Machine UNIX Programmer's Manual

**Volume 1, Sections 2-8**

**TI-2242803-0001**

**November, 1983**

Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA

## NAME

intro, errno - introduction to system calls and error numbers

## SYNOPSIS

**#include <errno.h>**

## DESCRIPTION

Section 2 of this manual lists all the entries into the system. Most of these calls have an error return. An error condition is indicated by an otherwise impossible returned value. Almost always this is -1; the individual sections specify the details. An error number is also made available in the external variable errno. Errno is not cleared on successful calls, so it should be tested only after an error has occurred.

There is a table of messages associated with each error, and a routine for printing the message; See perror(3). The possible error numbers are not recited with each writeup in section 2, since many errors are possible for most of the calls. Here is a list of the error numbers, their names as defined in <errno.h>, and the messages available using perror.

Ø Error Ø
Unused.

1 EPERM Not owner
Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory
This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process
The process whose number was given to signal and ptrace does not exist, or is already dead.

4 EINTR Interrupted system call
An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error

Some physical I/O error occurred during a read or write. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
I/O on a special file refers to a subdevice that does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not dialled in or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 5120 bytes is presented to exec.

8 ENOEXEC Exec format error
A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number, see a.out(5).

9 EBADF Bad file number
Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file that is open only for writing (resp. reading).

10 ECHILD No children
Wait and the process has no living or unwaited-for children.

11 EAGAIN No more processes
In a fork, the system's process table is full or the user is not allowed to create any more processes.

12 ENOMEM Not enough core
During an exec or break, a program asks for more core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers.

13 EACCES Permission denied
An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address
The system encountered a hardware fault in attempting to access the arguments of a system call.

15 ENOTBLK Block device required
A plain file was mentioned where a block device was required, e.g. in mount.

16 EBUSY Mount device busy
An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment).

17 EEXIST File exists
An existing file was mentioned in an inappropriate context, e.g. link.

18 EXDEV Cross-device link
A link to a file on another device was attempted.

19 ENODEV No such device
An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20 ENOTDIR Not a directory
A non-directory was specified where a directory is required, for example in a path name or as an argument to chdir.

21 EISDIR Is a directory
An attempt to write on a directory.

22 EINVAL Invalid argument
Some invalid argument: dismounting a non-mounted device, mentioning an unknown signal in signal, reading or writing a file for which seek has generated a negative pointer. Also set by math functions, see intro(3).

23 ENFILE File table overflow
The system's table of open files is full, and temporarily no more opens can be accepted.

24 EMFILE Too many open files
Customary configuration limit is 20 per process.

25 ENOTTY Not a typewriter
The file mentioned in stty or gtty is not a terminal or one of the other devices to which these calls apply.

26 ETXTBSY Text file busy
An attempt to execute a pure-procedure program that is currently open for writing (or reading!). Also an attempt to open for writing a pure-procedure program that is being executed.

27 EFBIG File too large
The size of a file exceeded the maximum (about 1.0E9 bytes).

28 ENOSPC No space left on device
During a write to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek
An lseek was issued to a pipe. This error should also be issued for other non-seekable devices.

30 EROFS Read-only file system
An attempt to modify a file or directory was made on a device mounted read-only.

31 EMLINK Too many links
An attempt to make more than 32767 links to a file.

32 EPIPE Broken pipe
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument
The argument of a function in the math package (3M) is out of the domain of the function.

34 ERANGE Result too large
The value of a function in the math package (3M) is unrepresentable within machine precision.

**SEE ALSO**

intro(3)

**ASSEMBLER**

**as /usr/include/sys.s file ...**

The PDP11 assembly language interface is given for each system call. The assembler symbols are defined in `/usr/include/sys.s'.

Return values appear in registers r0 and r1; it is unwise to count on these registers being preserved when no value is expected. An erroneous call is always indicated by turning on the c-bit of the condition codes. The error number is returned in r0. The presence of an error is most easily tested by the instructions bes and bec (`branch on error set (or clear)'). These are synonyms for the bcs and bcc instructions.

On the Interdata 8/32, the system call arguments correspond well to the arguments of the C routines. The sequence is:

```
la   %2,errno
```

```
1     %Ø,&callno
svc   Ø,args
```

Thus register 2 points to a word into which the error number will be stored as needed; it is cleared if no error occurs. Register Ø contains the system call number; the nomenclature is identical to that on the PDP11. The argument of the svc is the address of the arguments, laid out in storage as in the C calling sequence. The return value is in register 2 (possibly 3 also, as in pipe) and is -1 in case of error. The overflow bit in the program status word is also set when errors occur.

**NAME**

access - determine accessibility of file

**SYNOPSIS**

```
access(name, mode)
char *name;
```

**DESCRIPTION**

Access checks the given file name for accessibility according to mode, which is 4 (read), 2 (write) or 1 (execute) or a combination thereof. Specifying mode Ø tests whether the directories leading to the file can be searched and the file exists.

An appropriate error indication is returned if name cannot be found or if any of the desired access modes would not be granted. On disallowed accesses -1 is returned and the error code is in errno. Ø is returned from successful tests.

The user and group IDs with respect to which permission is checked are the real UID and GID of the process, so this call is useful to set-UID programs.

Notice that it is only access bits that are checked. A directory may be announced as writable by access, but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but exec will fail unless it is in proper format.

**SEE ALSO**

stat(2)

**ASSEMBLER**

```
(access = 33.)
sys access; name; mode
```

## NAME

acct - turn accounting on or off

## SYNOPSIS

```
acct(file)
char *file;
```

## DESCRIPTION

The system is prepared to write a record in an accounting file for each process as it terminates. This call, with a null-terminated string naming an existing file as argument, turns on accounting; records for each terminating process are appended to file. An argument of Ø causes accounting to be turned off.

The accounting file format is given in acct(5).

## SEE ALSO

acct(5), sa(1)

## DIAGNOSTICS

On error -1 is returned. The file must exist and the call may be exercised only by the super-user. It is erroneous to try to turn on accounting when it is already on.

## BUGS

No accounting is produced for programs running when a crash occurs. In particular nonterminating programs are never accounted for.

## ASSEMBLER

```
(acct = 51.)
sys acct; file
```

## NAME

alarm - schedule signal after specified time

## SYNOPSIS

```
alarm(seconds)
unsigned seconds;
```

## DESCRIPTION

Alarm causes signal SIGALRM, see signal(2), to be sent to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is Ø, any alarm request is cancelled. Because the clock has a 1-second resolution, the signal may occur up to one second early; because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 65535 seconds.

The return value is the amount of time previously remaining in the alarm clock.

## SEE ALSO

pause(2), signal(2), sleep(3)

## ASSEMBLER

```
(alarm = 27.)
(seconds in rØ)
sys alarm
(previous amount in rØ)
```

## NAME

brk, sbrk, break - change core allocation

## SYNOPSIS

```
char *brk(addr)

char *sbrk(incr)
```

## DESCRIPTION

Brk sets the system's idea of the lowest location not used by the program (called the break) to addr (rounded up to the next multiple of 64 bytes on the PDP11, 256 bytes on the Interdata 8/32, 512 bytes on the VAX-11/780). Locations not less than addr and below the stack pointer are not in the address space and will thus cause a memory violation if accessed.

In the alternate function sbrk, incr more bytes are added to the program's data space and a pointer to the start of the new area is returned.

When a program begins execution via exec the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use break.

## SEE ALSO

exec(2), malloc(3), end(3)

## DIAGNOSTICS

Zero is returned if the break could be set; -1 if the program requests more memory than the system limit or if too many segmentation registers would be required to implement the break.

## BUGS

Setting the break in the range 0177701 to 0177777 (on the PDP11) is the same as setting it to zero.

## ASSEMBLER

(break = 17.)
**sys break; addr**

Break performs the function of brk. The name of the routine differs from that in C for historical reasons.

## NAME

chdir, chroot - change default directory

## SYNOPSIS

```
chdir(dirname)
char *dirname;

chroot(dirname)
char *dirname;
```

## DESCRIPTION

Dirname is the address of the pathname of a directory, terminated by a null byte. Chdir causes this directory to become the current working directory, the starting point for path names not beginning with `/'.

Chroot sets the root directory, the starting point for path names beginning with `/'. The call is restricted to the super-user.

## SEE ALSO

cd(1)

## DIAGNOSTICS

Zero is returned if the directory is changed; -1 is returned if the given name is not that of a directory or is not searchable.

## ASSEMBLER

```
(chdir = 12.)
sys chdir; dirname

(chroot = 61.)
sys chroot; dirname
```

**NAME**

chmod - change mode of file

**SYNOPSIS**

```
chmod(name, mode)
char *name;
```

**DESCRIPTION**

The file whose name is given as the null-terminated string pointed to by name has its mode changed to mode. Modes are constructed by ORing together some combination of the following:

```
04000 set user ID on execution
02000 set group ID on execution
01000 save text image after execution
00400 read by owner
00200 write by owner
00100 execute (search on directory) by owner
00070 read, write, execute (search) by group
00007 read, write, execute (search) by others
```

If an executable file is set up for sharing (**-n** or **-i** option of ld(1)) then mode 1000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time. Ability to set this bit is restricted to the super-user since swap space is consumed by the images; it is only worth while for heavily used commands.

Only the owner of a file (or the super-user) may change the mode. Only the super-user can set the 1000 mode.

**SEE ALSO**

chmod(1)

**DIAGNOSTIC**

Zero is returned if the mode is changed; -1 is returned if name cannot be found or if current user is neither the owner of the file nor the super-user.

**ASSEMBLER**

(chmod = 15.)

```
sys chmod; name; mode
```

## NAME

chown - change owner and group of a file

## SYNOPSIS

```
chown(name, owner, group)
char *name;
```

## DESCRIPTION

The file whose name is given by the null-terminated string pointed to by name has its owner and group changed as specified. Only the super-user may execute this call, because if users were able to give files away, they could defeat the (nonexistent) file-space accounting procedures.

## SEE ALSO

chown(1), passwd(5)

## DIAGNOSTICS

Zero is returned if the owner is changed; -1 is returned on illegal owner changes.

## ASSEMBLER

(chown = 16.)

```
sys chown; name; owner; group
```

## NAME

close - close a file

## SYNOPSIS

**close(fildes)**

## DESCRIPTION

Given a file descriptor such as returned from an open, creat, dup or pipe(2) call, close closes the associated file. A close of all files is automatic on exit, but since there is a limit on the number of open files per process, close is necessary for programs which deal with many files.

Files are closed upon termination of a process, and certain file descriptors may be closed by exec(2) (see ioctl(2)).

## SEE ALSO

creat(2), open(2), pipe(2), exec(2), ioctl(2)

## DIAGNOSTICS

Zero is returned if a file is closed; -1 is returned for an unknown file descriptor.

## ASSEMBLER

(close = 6.)
(file descriptor in rØ)
**sys close**

## NAME

creat - create a new file

## SYNOPSIS

```
creat(name, mode)
char *name;
```

## DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file called name, given as the address of a null-terminated string. If the file did not exist, it is given mode mode, as modified by the process's mode mask (see umask(2)). Also see chmod(2) for the construction of the mode argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to Ø length.

The file is also opened for writing, and its file descriptor is returned.

The mode given is arbitrary; it need not allow writing. This feature is used by programs which deal with temporary files of fixed names. The creation is done with a mode that forbids writing. Then if a second instance of the program attempts a creat, an error is returned and the program knows that the name is unusable for the moment.

## SEE ALSO

write(2), close(2), chmod(2), umask (2)

## DIAGNOSTICS

The value -1 is returned if: a needed directory is not searchable; the file does not exist and the directory in which it is to be created is not writable; the file does exist and is unwritable; the file is a directory; there are already too many files open.

## ASSEMBLER

```
(creat = 8.)
sys creat; name; mode
(file descriptor in rØ)
```

## NAME

dup, dup2 - duplicate an open file descriptor

## SYNOPSIS

```
dup(fildes)
int fildes;

dup2(fildes, fildes2)
int fildes, fildes2;
```

## DESCRIPTION

Given a file descriptor returned from an open, pipe, or creat call, dup allocates another file descriptor synonymous with the original. The new file descriptor is returned.

In the second form of the call, fildes is a file descriptor referring to an open file, and fildes2 is a non-negative integer less than the maximum value allowed for file descriptors (approximately 19). Dup2 causes fildes2 to refer to the same file as fildes. If fildes2 already referred to an open file, it is closed first.

## SEE ALSO

creat(2), open(2), close(2), pipe(2)

## DIAGNOSTICS

The value -1 is returned if: the given file descriptor is invalid; there are already too many open files.

## ASSEMBLER

```
(dup = 41.)
(file descriptor in rØ)
(new file descriptor in r1)
sys dup
(file descriptor in rØ)
```

The dup2 entry is implemented by adding Ø1ØØ to fildes.

## NAME

execl, execv, execle, execve, execlp, execvp, exec, exece, environ - execute a file

## SYNOPSIS

```
execl(name, arg0, argl, ..., argn, 0)
char *name, *arg0, *argl, ..., *argn;

execv(name, argv)
char *name, *argv[ ];

execle(name, arg0, argl, ..., argn, 0, envp)
char *name, *arg0, *argl, ..., *argn, *envp[ ];

execve(name, argv, envp);
char *name, *argv[ ], *envp[ ];

extern char **environ;
```

## DESCRIPTION

Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

Files remain open across exec unless explicit arrangement has been made; see ioctl(2). Ignored signals remain ignored across these calls, but signals that are caught (see signal(2)) are reset to their default values.

Each user has a real user ID and group ID and an effective user ID and group ID. The real ID identifies the person using the system; the effective ID determines his access privileges. Exec changes the effective user and group ID to the owner of the executed file if the file has the `set-user-ID' or `set-group-ID' modes. The real user ID is not affected.

The name argument is a pointer to the name of the file to be executed. The pointers arg[0], arg[1] ... address null-terminated strings. Conventionally arg[0] is the name of the file.

From C, two interfaces are available. Execl is useful when a known file with known arguments is being called; the arguments to execl are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component). A 0 argument must end the argument list.

The execv version is useful when the number of arguments is unknown in advance; the arguments to execv are the name of

the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a Ø pointer.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where argc is the argument count and argv is an array of character pointers to the arguments themselves. As indicated, argc is conventionally at least one and the first member of the array points to a string containing the name of the file.

Argv is directly usable in another execv because argv[argc] is Ø.

Envp is a pointer to an array of strings that constitute the environment of the process. Each string consists of a name, an ``='', and a null-terminated value. The array of pointers is terminated by a null pointer. The shell sh(1) passes an environment entry for each global shell variable defined when the program is called. See environ(5) for some conventionally used names. The C run-time start-off routine places a copy of envp in the global cell environ, which is used by execv and execl to pass the environment to any subprograms executed by the current program. The exec routines use lower-level routines as follows to pass an environment explicitly:

```
execle(file, argØ, argl, . . . , argn, Ø, environ);
execve(file, argv, environ);
```

Execlp and execvp are called with the same arguments as execl and execv, but duplicate the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

**FILES**

/bin/sh shell, invoked if command file found by execlp or execvp

**SEE ALSO**

fork(2), environ(5)

**DIAGNOSTICS**

If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see a.out(5)), if maximum memory is exceeded, or if the arguments require too much space, a return constitutes the diagnostic; the return value is -1. Even for the super-user, at least one of the

execute-permission bits must be set for a file to be executed.

## BUGS

If execvp is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of argv[Ø] and argv[-1] will be modified before return.

## ASSEMBLER

(exec = 11.)
**sys exec; name; argv**

(exece = 59.)
**sys exece; name; argv; envp**

Plain exec is obsoleted by exece, but remains for historical reasons.

When the called file starts execution on the PDP11, the stack pointer points to a word containing the number of arguments. Just above this number is a list of pointers to the argument strings, followed by a null pointer, followed by the pointers to the environment strings and then another null pointer. The strings themselves follow; a Ø word is left at the very top of memory.

```
  sp->    nargs
     argØ
     ...
     argn
     Ø
     envØ
     ...
     envm
     Ø

 argØ:    <argØ\Ø>
     ...
 envØ:    <envØ\Ø>
     Ø
```

On the Interdata 8/32, the stack begins at a conventional place (currently ØxDØØØØ) and grows upwards. After exec, the layout of data on the stack is as follows.

```
     int  Ø
 argØ:    byte ...
     ...
argpØ:    int  argØ
     ...
     int  Ø
```

```
envpØ:    int  envØ
     ...
     int  Ø
 %2->      space     4Ø
     int  nargs
     int  argpØ
     int  envpØ
 %3->
```

This arrangement happens to conform well to C calling conventions.

## NAME

exit - terminate process

## SYNOPSIS

```
exit(status)
int status;

_exit(status)
int status;
```

## DESCRIPTION

Exit is the normal means of terminating a process. Exit closes all the process's files and notifies the parent process if it is executing a wait. The low-order 8 bits of status are available to the parent process.

This call can never return.

The C function exit may cause cleanup actions before the final `sys exit'. The function _exit circumvents all cleanup.

## SEE ALSO

wait(2)

## ASSEMBLER

```
(exit = 1.)
(status in rØ)
sys exit
```

## NAME

fork - spawn new process

## SYNOPSIS

**fork( )**

## DESCRIPTION

Fork is the only way new processes are created. The new process's core image is a copy of that of the caller of fork. The only distinction is the fact that the value returned in the old (parent) process contains the process ID of the new (child) process, while the value returned in the child is Ø. Process ID's range from 1 to 3Ø,ØØØ. This process ID is used by wait(2).

Files open before the fork are shared, and have a common read-write pointer. In particular, this is the way that standard input and output files are passed and also how pipes are set up.

## SEE ALSO

wait(2), exec(2)

## DIAGNOSTICS

Returns -1 and fails to create a process if: there is inadequate swap space, the user is not super-user and has too many processes, or the system's process table is full. Only the super-user can take the last process-table slot.

## ASSEMBLER

(fork = 2.)
**sys fork**
(new process return)
(old process return, new process ID in rØ)

The return locations in the old and new process differ by one word. The C-bit is set in the old process if a new process could not be created.

**NAME**

getpid - get process identification

**SYNOPSIS**

**getpid( )**

**DESCRIPTION**

Getpid returns the process ID of the current process. Most often it is used to generate uniquely-named temporary files.

**SEE ALSO**

mktemp(3)

**ASSEMBLER**

```
(getpid = 20.)
sys getpid
(pid in r0)
```

## NAME

getuid, getgid, geteuid, getegid - get user and group identity

## SYNOPSIS

```
getuid( )

geteuid( )

getgid( )

getegid( )
```

## DESCRIPTION

Getuid returns the real user ID of the current process, geteuid the effective user ID. The real user ID identifies the person who is logged in, in contradistinction to the effective user ID, which determines his access permission at the moment. It is thus useful to programs which operate using the `set user ID' mode, to find out who invoked them.

Getgid returns the real group ID, getegid the effective group ID.

## SEE ALSO

setuid(2)

## ASSEMBLER

```
(getuid = 24.)
sys getuid
(real user ID in rØ, effective user ID in r1)

(getgid = 47.)
sys getgid
(real group ID in rØ, effective group ID in r1)
```

## NAME

ioctl, stty, gtty - control device

## SYNOPSIS

```
#include <sgtty.h>

ioctl(fildes, request, argp)
struct sgttyb *argp;

stty(fildes, argp)
struct sgttyb *argp;

gtty(fildes, argp)
struct sgttyb *argp;
```

## DESCRIPTION

Ioctl performs a variety of functions on character special files (devices). The writeups of various devices in section 4 discuss how ioctl applies to them.

For certain status setting and status inquiries about terminal devices, the functions stty and gtty are equivalent to

```
ioctl(fildes, TIOCSETP, argp)
ioctl(fildes, TIOCGETP, argp)
```

respectively; see tty(4).

The following two calls, however, apply to any open file:

```
ioctl(fildes, FIOCLEX, NULL);
ioctl(fildes, FIONCLEX, NULL);
```

The first causes the file to be closed automatically during a successful exec operation; the second reverses the effect of the first.

## SEE ALSO

stty(1), tty(4), exec(2)

## DIAGNOSTICS

Zero is returned if the call was successful; -1 if the file descriptor does not refer to the kind of file for which it was intended.

## BUGS

Strictly speaking, since ioctl may be extended in different ways to devices with different properties, argp should have an open-ended declaration like

```
union { struct sgttyb ...; ... } *argp;
```

The important thing is that the size is fixed by `struct sgttyb'.

**ASSEMBLER**

```
(ioctl = 54.)
sys ioctl; fildes; request; argp

(stty = 31.)
(file descriptor in r0)
stty; argp

(gtty = 32.)
(file descriptor in r0)
sys gtty; argp
```

## NAME

kill - send signal to a process

## SYNOPSIS

**kill(pid, sig);**

## DESCRIPTION

Kill sends the signal sig to the process specified by the process number in rØ. See signal(2) for a list of signals.

The sending and receiving processes must have the same effective user ID, otherwise this call is restricted to the super-user.

If the process number is Ø, the signal is sent to all other processes in the sender's process group; see tty(4).

If the process number is -1, and the user is the super-user, the signal is broadcast universally except to processes Ø and 1, the scheduler and initialization processes, see init(8).

Processes may send signals to themselves.

## SEE ALSO

signal(2), kill(1)

## DIAGNOSTICS

Zero is returned if the process is killed; -1 is returned if the process does not have the same effective user ID and the user is not super-user, or if the process does not exist.

## ASSEMBLER

(kill = 37.)
(process number in rØ)
**sys kill; sig**

## NAME

link - link to a file

## SYNOPSIS

```
link(name1, name2)
char *name1, *name2;
```

## DESCRIPTION

A link to name1 is created; the link has the name name2. Either name may be an arbitrary path name.

## SEE ALSO

ln(1), unlink(2)

## DIAGNOSTICS

Zero is returned when a link is made; -1 is returned when name1 cannot be found; when name2 already exists; when the directory of name2 cannot be written; when an attempt is made to link to a directory by a user other than the super-user; when an attempt is made to link to a file on another file system; when a file has too many links.

## ASSEMBLER

```
(link = 9.)
sys link; name1; name2
```

## NAME

lock - lock a process in primary memory

## SYNOPSIS

**lock(flag)**

## DESCRIPTION

If the flag argument is non-zero, the process executing this call will not be swapped except if it is required to grow. If the argument is zero, the process is unlocked. This call may only be executed by the super-user.

## BUGS

Locked processes interfere with the compaction of primary memory and can cause deadlock. This system call is not considered a permanent part of the system.

## ASSEMBLER

(lock = 53.)
**sys lock; flag**

NAME

lseek, tell - move read/write pointer

SYNOPSIS

**long lseek(fildes, offset, whence)**
**long offset;**

**long tell(fildes)**

DESCRIPTION

The file descriptor refers to a file open for reading or writing. The read (resp. write) pointer for the file is set as follows:

If whence is Ø, the pointer is set to offset bytes.

If whence is 1, the pointer is set to its current location plus offset.

If whence is 2, the pointer is set to the size of the file plus offset.

The returned value is the resulting pointer location.

The obsolete function tell(fildes) is identical to lseek(fildes, ØL, 1).

Seeking far beyond the end of a file, then writing, creates a gap or `hole', which occupies no physical space and reads as zeros.

SEE ALSO

open(2), creat(2), fseek(3)

DIAGNOSTICS

-1 is returned for an undefined file descriptor, seek on a pipe, or seek to a position before the beginning of file.

BUGS

Lseek is a no-op on character special files.

ASSEMBLER

(lseek = 19.)
(file descriptor in rØ)
**sys lseek; offset1; offset2; whence**

Offset1 and offset2 are the high and low words of offset; rØ and r1 contain the pointer upon return.

## NAME

mknod - make a directory or a special file

## SYNOPSIS

**mknod(name, mode, addr)**
**char *name;**

## DESCRIPTION

Mknod creates a new file whose name is the null-terminated string pointed to by name. The mode of the new file (including directory and special file bits) is initialized from mode. (The protection part of the mode is modified by the process's mode mask; see umask(2)). The first block pointer of the i-node is initialized from addr. For ordinary files and directories addr is normally zero. In the case of a special file, addr specifies which special file.

Mknod may be invoked only by the super-user.

## SEE ALSO

mkdir(1), mknod(1), filsys(5)

## DIAGNOSTICS

Zero is returned if the file has been made; -1 if the file already exists or if the user is not the super-user.

## ASSEMBLER

(mknod = 14.)
**sys mknod; name; mode; addr**

## NAME

mount, umount - mount or remove file system

## SYNOPSIS

```
mount(special, name, rwflag)
char *special, *name;

umount(special)
char *special;
```

## DESCRIPTION

Mount announces to the system that a removable file system has been mounted on the block-structured special file special; from now on, references to file name will refer to the root file on the newly mounted file system. Special and name are pointers to null-terminated strings containing the appropriate path names.

Name must exist already. Name must be a directory (unless the root of the mounted file system is not a directory). Its old contents are inaccessible while the file system is mounted.

The rwflag argument determines whether the file system can be written on; if it is Ø writing is allowed, if non-zero no writing is done. Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the special file is no longer to contain a removable file system. The associated file reverts to its ordinary interpretation.

## SEE ALSO

mount(1)

## DIAGNOSTICS

Mount returns Ø if the action occurred; -1 if special is inaccessible or not an appropriate file; if name does not exist; if special is already mounted; if name is in use; or if there are already too many file systems mounted.

Umount returns Ø if the action occurred; -1 if if the special file is inaccessible or does not have a mounted file system, or if there are active files in the mounted file system.

## ASSEMBLER

(mount = 21.)

```
sys mount; special; name; rwflag
```

(umount = 22.)
**sys umount; special**

## NAME

mpxcall - multiplexor and channel interface

## SYNOPSIS

**mpxcall(arg1, arg2, arg3, cmd)**

## DESCRIPTION

Mpxcall supplies a primitive interface to the kernel used by the routines listed below. Each routine that uses mpxcall passes an integer cmd as the fourth argument. These are defined in /usr/include/mx.h. Mpxcall always returns an integer which is to be interpreted in accordance with the definition of cmd.

## SEE ALSO

group(2), join(2), extract(2), connect(2), chan(2), attach(2), detach(2)

## DIAGNOSTICS

The value -1 is returned on error.

## NAME

nice - change priority of a process

## SYNOPSIS

```
int nice (incr)
int incr;
```

## DESCRIPTION

Nice adds the value of incr to the nice value of the calling process. A process's nice value is a positive number for which a more positive value results in lower CPU priority.

A maximum nice value of 39 and a minimum nice value of Ø are imposed by the system. Requests for values above or below these limits result in the nice value being set to the corresponding limit.

Nice will fail and not change the nice value if incr is negative and the effective user ID of the calling process is not super-user. [EPERM]

## RETURN VALUE

Upon successful completion, nice returns the new nice value minus 2Ø. Otherwise, a value of -1 is returned and errno is set to indicate the error.

## SEE ALSO

nice(1), exec(2).

## NAME

open - open for reading or writing

## SYNOPSIS

```
open(name, mode)
char *name;
```

## DESCRIPTION

Open opens the file name for reading (if mode is Ø), writing (if mode is 1) or for both reading and writing (if mode is 2). Name is the address of a string of ASCII characters representing a path name, terminated by a null character.

The file is positioned at the beginning (byte Ø). The returned file descriptor must be used for subsequent calls for other input-output functions on the file.

## SEE ALSO

creat(2), read(2), write(2), dup(2), close(2)

## DIAGNOSTICS

The value -1 is returned if the file does not exist, if one of the necessary directories does not exist or is unreadable, if the file is not readable (resp. writable), or if too many files are open.

## ASSEMBLER

```
(open = 5.)
sys open; name; mode
(file descriptor in rØ)
```

**NAME**

pause - stop until signal

**SYNOPSIS**

**pause( )**

**DESCRIPTION**

Pause never returns normally. It is used to give up control while waiting for a signal from kill(2) or alarm(2).

**SEE ALSO**

kill(1), kill(2), alarm(2), signal(2), setjmp(3)

**ASSEMBLER**

(pause = 29.)
**sys pause**

## NAME

phys - allow a process to access physical addresses

## SYNOPSIS

**phys(segreg, size, physadr)**

## DESCRIPTION

The argument segreg specifies a process virtual (data-space) address range of 8K bytes starting at virtual address segregx8K bytes. This address range is mapped into physical address physadrx64 bytes. Only the first sizex64 bytes of this mapping is addressable. If size is zero, any previous mapping of this virtual address range is nullified. For example, the call

```
phys(6, 1, Ø177775);
```

will map virtual addresses Ø16ØØØØ-Ø16ØØ77 into physical addresses Ø17777ØØ-Ø17777577. In particular, virtual address Ø16ØØ6Ø is the PDP-11 console located at physical address Ø17777560.

This call may only be executed by the super-user.

## SEE ALSO

PDP-11 segmentation hardware

## DIAGNOSTICS

The function value zero is returned if the physical mapping is in effect. The value -1 is returned if not super-user, if segreg is not in the range Ø-7, if size is not in the range Ø-127, or if the specified segreg is already used for other than a previous call to phys.

## BUGS

This system call is obviously very machine dependent and very dangerous. This system call is not considered a permanent part of the system.

## ASSEMBLER

(phys = 52.)
**sys phys; segreg; size; physadr**

## NAME

pipe - create an interprocess channel

## SYNOPSIS

```
pipe(fildes)
int fildes[2];
```

## DESCRIPTION

The pipe system call creates an I/O mechanism called a pipe. The file descriptors returned can be used in read and write operations. When the pipe is written using the descriptor fildes[1] up to 4096 bytes of data are buffered before the writing process is suspended. A read using the descriptor fildes[0] will pick up the data. Writes with a count of 4096 bytes or less are atomic; no other process can intersperse data.

It is assumed that after the pipe has been set up, two (or more) cooperating processes (created by subsequent fork calls) will pass data through the pipe with read and write calls.

The Shell has a syntax to set up a linear array of processes connected by pipes.

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) returns an end-of-file.

## SEE ALSO

sh(1), read(2), write(2), fork(2)

## DIAGNOSTICS

The function value zero is returned if the pipe was created; -1 if too many files are already open. A signal is generated if a write on a pipe with only one end is attempted.

## BUGS

Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.

## ASSEMBLER

```
(pipe = 42.)
sys pipe
(read file descriptor in r0)
(write file descriptor in r1)
```

## NAME

profil - execution time profile

## SYNOPSIS

```
profil(buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;
```

## DESCRIPTION

Buff points to an area of core whose length (in bytes) is given by bufsiz. After this call, the user's program counter (pc) is examined each clock tick (60th second); offset is subtracted from it, and the result multiplied by scale. If the resulting number corresponds to a word inside buff, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777(8) gives a 1-1 mapping of pc's to words in buff; 077777(8) maps each pair of instruction words together. 02(8) maps all instructions onto the beginning of buff (producing a non-interrupting core clock).

Profiling is turned off by giving a scale of 0 or 1. It is rendered ineffective by giving a bufsiz of 0. Profiling is turned off when an exec is executed, but remains on in child and parent both after a fork. Profiling may be turned off if an update in buff would cause a memory fault.

## SEE ALSO

monitor(3), prof(1)

## ASSEMBLER

(profil = 44.)
**sys profil; buff; bufsiz; offset; scale**

NAME

ptrace - process trace

SYNOPSIS

```
#include <signal.h>

ptrace(request, pid, addr, data)
int *addr;
```

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging. There are four arguments whose interpretation depends on a request argument. Generally, pid is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like `illegal instruction' or externally generated like `interrupt.' See signal(2) for the list. Then the traced process enters a stopped state and its parent is notified via wait(2). When the child is in the stopped state, its core image can be examined and modified using ptrace. If desired, another ptrace request can then cause the child either to terminate or to continue, possibly ignoring the signal.

The value of the request argument determines the precise action of the call:

Ø This request is the only one used by the child process; it declares that the process is to be traced by its parent. All the other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

1,2 The word in the child process's address space at addr is returned. If I and D space are separated, request 1 indicates I space, 2 D space. Addr must be even. The child must be stopped. The input data is ignored.

3 The word of the system's per-process data area corresponding to addr is returned. Addr must be even and less than 512. This space contains the registers and other information about the process; its layout corresponds to the user structure in the system.

4,5 The given data is written at the word in the process's address space corresponding to addr, which must be even. No useful value is returned. If I and D space are separated, request 4 indicates I space, 5 D space.

Attempts to write in pure procedure fail if another process is executing the same file.

6 The process's system data is written, as it is read with request 3. Only a few locations can be written in this way: the general registers, the floating point status and registers, and certain bits of the processor status word.

7 The data argument is taken as a signal number and the child's execution continues at location addr as if it had incurred that signal. Normally the signal number will be either Ø to indicate that the signal that caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop. If addr is (int *)1 then execution continues from where it stopped.

8 The traced process terminates.

9 Execution continues as in request 7; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is SIGTRAP. (On the PDP-11 the T-bit is used and just one instruction is executed; on the Interdata the stop does not take place until a store instruction is executed.) This is part of the mechanism for implementing breakpoints.

As indicated, these calls (except for request Ø) can be used only when the subject process has stopped. The wait call is used to determine when a process stops; in such a case the `termination' status returned by wait has the value Ø177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, ptrace inhibits the set-user-id facility on subsequent exec(2) calls. If a traced process calls exec, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

On the Interdata 8/32, `word' means a 32-bit word and `even' means Ø mod 4.

**SEE ALSO**

wait(2), signal(2), adb(1)

**DIAGNOSTICS**

The value -1 is returned if request is invalid, pid is not a traceable process, addr is out of bounds, or data specifies an illegal signal number.

**BUGS**

On the Interdata 8/32, `as soon as possible' (request 7) means `as soon as a store instruction has been executed.'

The request Ø call should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use `illegal instruction' signals at a very high rate) could be efficiently debugged.
The error indication, -1, is a legitimate function value; errno, see intro(2), can be used to disambiguate.

It should be possible to stop a process on occurrence of a system call; in this way a completely controlled environment could be provided.

**ASSEMBLER**

(ptrace = 26.)
(data in rØ)
**sys ptrace; pid; addr; request**
(value in rØ)

## NAME

read - read from file

## SYNOPSIS

**read(fildes, buffer, nbytes)**
**char *buffer;**

## DESCRIPTION

A file descriptor is a word returned from a successful open, creat, dup, or pipe call. Buffer is the location of nbytes contiguous bytes into which the input will be placed. It is not guaranteed that all nbytes bytes will be read; for example if the file refers to a typewriter at most one line will be returned. In any event the number of characters read is returned.

If the returned value is Ø, then end-of-file has been reached.

## SEE ALSO

open(2), creat(2), dup(2), pipe(2)

## DIAGNOSTICS

As mentioned, Ø is returned when the end of the file has been reached. If the read was otherwise unsuccessful the return value is -1. Many conditions can generate an error: physical I/O errors, bad buffer address, preposterous nbytes, file descriptor not that of an input file.

## ASSEMBLER

(read = 3.)
(file descriptor in rØ)
**sys read; buffer; nbytes**
(byte count in rØ)

## NAME

setuid, setgid - set user and group ID

## SYNOPSIS

**setuid(uid)**

**setgid(gid)**

## DESCRIPTION

The user ID (group ID) of the current process is set to the argument. Both the effective and the real ID are set. These calls are only permitted to the super-user or if the argument is the real ID.

## SEE ALSO

getuid(2)

## DIAGNOSTICS

Zero is returned if the user (group) ID is set; -1 is returned otherwise.

## ASSEMBLER

(setuid = 23.)
(user ID in rØ)
**sys setuid**

(setgid = 46.)
(group ID in rØ)
**sys setgid**

NAME

signal - catch or ignore signals

SYNOPSIS

```
#include <signal.h>

(*signal(sig, func))()
(*func)();
```

DESCRIPTION

A signal is generated by some abnormal event, initiated either by user at a typewriter (quit, interrupt), by a program error (bus error, etc.), or by request of another program (kill). Normally all signals cause termination of the receiving process, but a signal call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals with names as in the include file.

| | | |
|---|---|---|
| SIGHUP | 1 | hangup |
| SIGINT | 2 | interrupt |
| SIGQUIT | 3* | quit |
| SIGILL | 4* | illegal instruction (not reset when caught) |
| SIGTRAP | 5* | trace trap (not reset when caught) |
| SIGIOT | 6* | IOT instruction |
| SIGEMT | 7* | EMT instruction |
| SIGFPE | 8* | floating point exception |
| SIGKILL | 9 | kill (cannot be caught or ignored) |
| SIGBUS | 10* | bus error |
| SIGSEGV | 11* | segmentation violation |
| SIGSYS | 12* | bad argument to system call |
| SIGPIPE | 13 | write on a pipe or link with no one to read it |
| SIGALRM | 14 | alarm clock |
| SIGTERM | 15 | software termination signal |
| | 16 | unassigned |

The starred signals in the list above cause a core image if not caught or ignored.

If func is SIG_DFL, the default action for signal sig is reinstated; this default is termination, sometimes with a core image. If func is SIG_IGN the signal is ignored. Otherwise when the signal occurs func will be called with the signal number as argument. A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal is reset to SIG_DFL after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another signal call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a read or write(2) on a slow device (like a

typewriter; but not a file); and during pause or wait(2). When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call.

The value of signal is the previous (or initial) value of func for the particular signal.

After a fork(2) the child inherits all signals. Exec(2) resets all caught signals to default action.

**SEE ALSO**

kill(1), kill(2), ptrace(2), setjmp(3)

**DIAGNOSTICS**

The value (int)-1 is returned if the given signal is out of range.

**BUGS**

If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

The type specification of the routine and its func argument are problematical.

**ASSEMBLER**

(signal = 48.)
**sys signal; sig; label**
(old label in rØ)

If label is Ø, default action is reinstated. If label is odd, the signal is ignored. Any other even label specifies an address in the process where an interrupt is simulated. An RTI or RTT instruction will return from the interrupt.

NAME

stat, fstat - get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

stat(name, buf)
char *name;
struct stat *buf;

fstat(fildes, buf)
struct stat *buf;
```

DESCRIPTION

Stat obtains detailed information about a named file. Fstat obtains the same information about an open file known by the file descriptor from a successful open, creat, dup or pipe(2) call.

Name points to a null-terminated string naming a file; buf is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be searchable. The layout of the structure pointed to by buf as defined in <stat.h> is given below. St mode is encoded according to the `#define' statements.

```
/*
     (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983. ALL
     RIGHTS RESERVED. PROPERTY OF TEXAS INSTRUMENTS INCORPORATED.
     RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE IS SUBJECT
     TO RESTRICTIONS SET FORTH IN TI'S PROGRAM LICENSE AGREEMENT AND
     ASSOCIATED DOCUMENTATION.
*/

#ifdef SCCSID
static char id_stat[] = "@(#)stat.h     1.2 (Texas Instruments) 83/06/
#endif

#ifndef SCCSID

struct    stat
{
     dev_t     st_dev;
     ino_t     st_ino;
     unsigned short      st_mode;
     short     st_nlink;
     short     st_uid;
     short     st_gid;
     dev_t     st_rdev;
```

```
        off_t     st_size;
        time_t    st_atime;
        time_t    st_mtime;
        time_t    st_ctime;
};

#define   S_IFMT    0170000        /* type of file */
#define        S_IFDIR   0040000   /* directory */
#define        S_IFCHR   0020000   /* character special */
#define        S_IFBLK   0060000   /* block special */
#define        S_IFREG   0100000   /* regular */
#define        S_IFMPC   0030000   /* multiplexed char special */
#define        S_IFMPB   0070000   /* multiplexed block special */
#define   S_ISUID   0004000        /* set user id on execution */
#define   S_ISGID   0002000        /* set group id on execution */
#define   S_ISVTX   0001000        /* save swapped text even after use
#define   S_IREAD   0000400        /* read permission, owner */
#define   S_IWRITE  0000200        /* write permission, owner */
#define   S_IEXEC   0000100        /* execute/search permission, owner

#endif
```

The mode bits 0000070 and 0000007 encode group and others permissions (see chmod(2)). The defined types, ino t, off t, time t, name various width integer values; dev t encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h> (see types(5).

When fildes is associated with a pipe, fstat reports an ordinary file with restricted permissions. The size is the number of bytes queued in the pipe.

st atime is the file was last read. For reasons of efficiency, it is not set when a directory is searched, although this would be more logical. st mtime is the time the file was last written or created. It is not set by changes of owner, group, link count, or mode. st ctime is set both both by writing and changing the i-node.

**SEE ALSO**

ls(1), filsys(5)

**DIAGNOSTICS**

Zero is returned if a status is available; -1 if the file cannot be found.

**ASSEMBLER**

(stat = 18.)
**sys stat; name; buf**

(fstat = 28.)

```
(file descriptor in rØ)
sys fstat; buf
```

## NAME

stime - set time

## SYNOPSIS

```
stime(tp)
long *tp;
```

## DESCRIPTION

Stime sets the system's idea of the time and date. Time, pointed to by tp, is measured in seconds from ØØØØ GMT Jan 1, 197Ø. Only the super-user may use this call.

## SEE ALSO

date(1), time(2), ctime(3)

## DIAGNOSTICS

Zero is returned if the time was set; -1 if user is not the super-user.

## ASSEMBLER

```
(stime = 25.)
(time in rØ-r1)
sys stime
```

## NAME

sync - update super-block

## SYNOPSIS

**sync( )**

## DESCRIPTION

Sync causes all information in core memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs which examine a file system, for example icheck, df, etc. It is mandatory before a boot.

## SEE ALSO

sync(1), update(8)

## BUGS

The writing, although scheduled, is not necessarily complete upon return from sync.

## ASSEMBLER

(sync = 36.)
**sys sync**

## NAME

time, ftime - get date and time

## SYNOPSIS

```
long time(0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;
```

## DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If tloc is nonnull, the return value is also stored in the place to which tloc points.

The ftime entry fills in a structure pointed to by its argument, as defined by <sys/timeb.h>:

```
/*
     (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983.  ALL
     RIGHTS RESERVED.  PROPERTY OF TEXAS INSTRUMENTS INCORPORATED.
     RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE IS SUBJECT
     TO RESTRICTIONS SET FORTH IN TI'S PROGRAM LICENSE AGREEMENT AND
     ASSOCIATED DOCUMENTATION.
*/

#ifdef SCCSID
static char id_timeb[] = "@(#)timeb.h1.2 (Texas Instruments) 83/06/27"
#endif

#ifndef SCCSID

/*
 * Structure returned by ftime system call
 */
struct timeb {
     time_t   time;
     unsigned short millitm;
     short    timezone;
     short    dstflag;
};

#endif
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local

timezone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

**SEE ALSO**

date(1), stime(2), ctime(3)

**ASSEMBLER**

(ftime = 35.)
**sys ftime; bufptr**

(time = 13.; obsolete call)
**sys time**
(time since 1970 in r0-r1)

## NAME

times - get process times

## SYNOPSIS

```
times(buffer)
struct tbuffer *buffer;
```

## DESCRIPTION

Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ=60 in North America.

After the call, the buffer will appear as follows:

```
struct tbuffer {
     long proc_user_time;
     long proc_system_time;
     long child_user_time;
     long child_system_time;
};
```

The children times are the sum of the children's process times and their children's times.

## SEE ALSO

time(1), time(2)

## ASSEMBLER

```
(times = 43.)
sys times; buffer
```

## NAME

umask - set file creation mode mask

## SYNOPSIS

**umask(complmode)**

## DESCRIPTION

Umask sets a mask used whenever a file is created by creat(2) or mknod(2): the actual mode (see chmod(2)) of the newly-created file is the logical and of the given mode and the complement of the argument. Only the low-order 9 bits of the mask (the protection bits) participate. In other words, the mask shows the bits to be turned off when files are created.

The previous value of the mask is returned by the call. The value is initially Ø (no restrictions). The mask is inherited by child processes.

## SEE ALSO

creat(2), mknod(2), chmod(2)

## ASSEMBLER

(umask = 6Ø.)
**sys umask; complmode**

## NAME

unlink - remove directory entry

## SYNOPSIS

```
unlink(name)
char *name;
```

## DESCRIPTION

Name points to a null-terminated string. Unlink removes the entry for the file pointed to by name from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

## SEE ALSO

rm(1), link(2)

## DIAGNOSTICS

Zero is normally returned; -1 indicates that the file does not exist, that its directory cannot be written, or that the file contains pure procedure text that is currently in use. Write permission is not required on the file itself. It is also illegal to unlink a directory (except for the super-user).

## ASSEMBLER

(unlink = 1Ø.)

```
sys unlink; name
```

NAME

utime - set file times

SYNOPSIS

```
#include <sys/types.h>
utime(file, timep)
char *file;
time_t timep[2];
```

DESCRIPTION

The utime call uses the `accessed' and `updated' times in that order from the timep vector to set the corresponding recorded times for file.

The caller must be the owner of the file or the super-user. The `inode-changed' time of the file is set to the current time.

SEE ALSO

stat (2)

ASSEMBLER

(utime = 30.)

```
sys utime; file; timep
```

## NAME

wait - wait for process to terminate

## SYNOPSIS

```
wait(status)
int *status;

wait(Ø)
```

## DESCRIPTION

Wait causes its caller to delay until a signal is received or one of its child processes terminates. If any child has died since the last wait, return is immediate; if there are no children, return is immediate with the error bit set (resp. with a value of -1 returned). The normal return yields the process ID of the terminated child. In the case of several children several wait calls are needed to learn of all the deaths.

If (int)status is nonzero, the high byte of the word pointed to receives the low byte of the argument of exit when the child terminated. The low byte receives the termination status of the process. See signal(2) for a list of termination statuses (signals); Ø status indicates normal termination. A special status (Ø177) is returned for a stopped process which has not terminated and can be restarted. See ptrace(2). If the Ø2ØØ bit of the termination status is set, a core image of the process was produced by the system.

If the parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

## SEE ALSO

exit(2), fork(2), signal(2)

## DIAGNOSTICS

Returns -1 if there are no children not previously waited for.

## ASSEMBLER

```
(wait = 7.)
sys wait
(process ID in rØ)
(status in r1)
```

The high byte of the status is the low byte of rØ in the child at termination.

## NAME

write - write on a file

## SYNOPSIS

```
write(fildes, buffer, nbytes)
char *buffer;
```

## DESCRIPTION

A file descriptor is a word returned from a successful open, creat, dup, or pipe(2) call.

Buffer is the address of nbytes contiguous bytes which are written on the output file. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Writes which are multiples of 512 characters long and begin on a 512-byte boundary in the file are more efficient than any others.

## SEE ALSO

creat(2), open(2), pipe(2)

## DIAGNOSTICS

Returns -1 on error: bad descriptor, buffer address, or count; physical I/O errors.

## ASSEMBLER

```
(write = 4.)
(file descriptor in rØ)
sys write; buffer; nbytes
(byte count in rØ)
```

**NAME**

intro - introduction to library functions

**SYNOPSIS**

```
#include <stdio.h>

#include <math.h>
```

**DESCRIPTION**

This section describes functions that may be found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in section 2. Functions are divided into various libraries distinguished by the section number at the top of the page:

(3) These functions, together with those of section 2 and those marked (3S), constitute library libc, which is automatically loaded by the C compiler cc(1) and the Fortran compiler f77(1). The link editor ld(1) searches this library under the `-lc' option. Declarations for some of these functions may be obtained from include files indicated on the appropriate pages.

(3M) These functions constitute the math library, libm. They are automatically loaded as needed by the Fortran compiler f77(1). The link editor searches this library under the `-lm' option. Declarations for these functions may be obtained from the include file <math.h>.

(3S) These functions constitute the `standard I/O package', see stdio(3). These functions are in the library libc already mentioned. Declarations for these functions may be obtained from the include file <stdio.h>.

(3X) Various specialized libraries have not been given distinctive captions. The files in which these libraries are found are named on the appropriate pages.

**FILES**

/lib/libc.a
/lib/libm.a, /usr/lib/libm.a (one or the other)

**SEE ALSO**

stdio(3), nm(1), ld(1), cc(1), f77(1), intro(2)

**DIAGNOSTICS**

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these

cases the external variable errno (see intro(2)) is set to the value EDOM or ERANGE. The values of EDOM and ERANGE are defined in the include file <math.h>.

## NAME

abort - generate fault

## DESCRIPTION

Abort executes a trap instruction. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

## SEE ALSO

adb(1), signal(2), exit(2)

## DIAGNOSTICS

Usually `trap - core dumped' from the shell.

NAME

abs - integer absolute value

SYNOPSIS

abs(i)

DESCRIPTION

Abs returns the absolute value of its integer operand.

SEE ALSO

floor(3) for fabs

BUGS

You get what the hardware gives on the largest negative integer.

## NAME

assert - program verification

## SYNOPSIS

```
#include <assert.h>

assert (expression)
```

## DESCRIPTION

Assert is a macro that indicates expression is expected to be true at this point in the program. It causes an exit(2) with a diagnostic comment on the standard output when expression is false (Ø). Compiling with the cc(1) option -DNDEBUG effectively deletes assert from the program.

## DIAGNOSTICS

`Assertion failed: file f line n.' F is the source file and n the source line number of the assert statement.

## NAME

atof, atoi, atol - convert ASCII to numbers

## SYNOPSIS

```
double atof(nptr)
char *nptr;

atoi(nptr)
char *nptr;

long atol(nptr)
char *nptr;
```

## DESCRIPTION

These functions convert a string pointed to by nptr to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional `e' or `E' followed by an optionally signed integer.

Atoi and atol recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

## SEE ALSO

scanf(3)

## BUGS

There are no provisions for overflow.

## NAME

crypt, setkey, encrypt - DES encryption

## SYNOPSIS

```
char *crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;
```

## DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to crypt is a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The salt string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of setkey is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the encrypt entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by setkey. If edflag is 0, the argument is encrypted; if non-zero, it is decrypted.

## SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

## BUGS

The return value points to static data whose content is overwritten by each call.

## NAME

ctime, localtime, gmtime, asctime, timezone - convert date and time to ASCII

## SYNOPSIS

```
char *ctime(clock)
long *clock;

#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

## DESCRIPTION

Ctime converts a time pointed to by clock such as returned by time(2) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 Ø1:Ø3:52 1973\n\Ø
```

Localtime and gmtime return pointers to structures containing the broken-down time. Localtime corrects for the time zone and possible daylight savings time; gmtime converts directly to GMT, which is the time UNIX uses. Asctime converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
struct tm { /* see ctime(3) */
        int  tm_sec;
        int  tm_min;
        int  tm_hour;
        int  tm_mday;
        int  tm_mon;
        int  tm_year;
        int  tm_wday;
        int  tm_yday;
        int  tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (Ø-11), day of week (Sunday =

Ø), year - 19ØØ, day of year (Ø-365), and a flag that is nonzero if daylight saving time is in effect.

When local time is called for, the program consults the system to determine the time zone and whether the standard U.S.A. daylight saving time adjustment is appropriate. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

Timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is Ø, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; e.g. in Afghanistan timezone(-(6Ø*4+3Ø), Ø) is appropriate because it is 4:3Ø ahead of GMT and the string GMT+4:3Ø is produced.

**SEE ALSO**

time(2)

**BUGS**

The return values point to static data whose content is overwritten by each call.

## NAME

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii - character classification

## SYNOPSIS

```
#include <ctype.h>

isalpha(c)

. . .
```

## DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. Isascii is defined on all integer values; the rest are defined only where isascii is true and on the single non-ASCII value EOF (see stdio(3)).

| | |
|---|---|
| isalpha | c is a letter |
| isupper | c is an upper case letter |
| islower | c is a lower case letter |
| isdigit | c is a digit |
| isalnum | c is an alphanumeric character |
| isspace | c is a space, tab, carriage return, newline, or formfeed |
| ispunct | c is a punctuation character (neither control nor alphanumeric) |
| isprint | c is a printing character, code 040(8) (space) through 0176 (tilde) |
| iscntrl | c is a delete character (0177) or ordinary control character (less than 040). |
| isascii | c is an ASCII character, code less than 0200 |

## SEE ALSO

ascii(7)

## NAME

curses - screen functions with ``optimal'' cursor motion

## SYNOPSIS

cc [ flags ] files -lcurses -ltermlib [ libraries ]

## DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the refresh() tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine initscr() must be called before any of the other routines that deal with windows and screens are used.

## SEE ALSO

Screen Updating and Cursor Movement Optimization: A Library Package, Ken Arnold,
termcap (5), stty (2), setenv (3), setenv (3),

## AUTHOR

Ken Arnold

## FUNCTIONS

| | |
|---|---|
| addch(ch) | add a character to stdscr |
| addstr(str) | add a string to stdscr |
| box(win,vert,hor) | draw a box around a window |
| crmode() | set cbreak mode |
| clear() | clear stdscr |
| clearok(scr,boolf) | set clear flag for scr |
| clrtobot() | clear to bottom on stdscr |
| clrtoeol() | clear to end of line on stdscr |
| delwin(win) | delete win |
| echo() | set echo mode |
| erase() | erase stdscr |
| getch() | get a char through stdscr |
| getstr(str) | get a string through stdscr |
| gettmode() | get tty modes |
| getyx(win,y,x) | get (y,x) co-ordinates |
| inch() | get char at current (y,x) co-ordinates |
| initscr() | initialize screens |
| leaveok(win,boolf) | set leave flag for win |
| longname(termbuf,name) | get long name from termbuf |
| move(y,x) | move to (y,x) on stdscr |
| mvcur(lasty,lastx,newy,newx) | actually move cursor |
| newwin(lines,cols,begin_y,begin_x) | create a new window |
| nl() | set newline mapping |
| nocrmode() | unset cbreak mode |
| noecho() | unset echo mode |
| nonl() | unset newline mapping |
| noraw() | unset raw mode |

| | |
|---|---|
| overlay(win1,win2) | overlay win1 on win2 |
| overwrite(win1,win2) | overwrite win1 on top of win2 |
| printw(fmt,arg1,arg2,...) | printf on stdscr |
| raw() | set raw mode |
| refresh() | make current screen look like stdscr |
| restty() | reset tty flags to stored value |
| savetty() | stored current tty flags |
| scanw(fmt,arg1,arg2,...) | scanf through stdscr |
| scroll(win) | scroll win one line |
| scrollok(win,boolf) | set scroll flag |
| setterm(name) | set term variables for name |
| unctrl(ch) | printable version of ch |
| waddch(win,ch) | add char to win |
| waddstr(win,str) | add string to win |
| wclear(win) | clear win |
| wclrtobot(win) | clear to bottom of win |
| wclrtoeol(win) | clear to end of line on win |
| werase(win) | erase win |
| wgetch(win) | get a char through win |
| wgetstr(win,str) | get a string through win |
| winch(win) | get char at current (y,x) in win |
| wmove(win,y,x) | set current (y,x) co-ordinates on win |
| wprintw(win,fmt,arg1,arg2,...) | printf on win |
| wrefresh(win) | make screen look like win |
| wscanw(win,fmt,arg1,arg2,...) | scanf through win |

## NAME

ecvt, fcvt, gcvt - output conversion

## SYNOPSIS

```
char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
char *buf;
```

## DESCRIPTION

Ecvt converts the value to a null-terminated string of ndigit ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through decpt (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by sign is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to ecvt, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by ndigits.

Gcvt converts the value to a null-terminated ASCII string in buf and returns a pointer to buf. It attempts to produce ndigit significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

## SEE ALSO

printf(3)

## BUGS

The return values point to static data whose content is overwritten by each call.

## NAME

end, etext, edata - last locations in program

## SYNOPSIS

```
extern end;
extern etext;
extern edata;
```

## DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of etext is the first address above the program text, edata above the initialized data region, and end above the uninitialized data region.

When execution begins, the program break coincides with end, but many functions reset the program break, among them the routines of brk(2), malloc(3), standard input/output (stdio(3)), the profile (-p) option of cc(1), etc. The current value of the program break is reliably returned by `sbrk(Ø)', see brk(2).

## SEE ALSO

brk(2), malloc(3)

## NAME

exp, log, log1Ø, pow, sqrt - exponential, logarithm, power, square root

## SYNOPSIS

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log1Ø(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

## DESCRIPTION

Exp returns the exponential function of x.

Log returns the natural logarithm of x; log1Ø returns the base 1Ø logarithm.

Pow returns x8y9.

Sqrt returns the square root of x.

## SEE ALSO

hypot(3), sinh(3), intro(2)

## DIAGNOSTICS

Exp and pow return a huge value when the correct value would overflow; errno is set to ERANGE. Pow returns Ø and sets errno to EDOM when the second argument is negative and non-integral and when both arguments are Ø.

Log returns Ø when x is zero or negative; errno is set to EDOM.

Sqrt returns Ø when x is negative; errno is set to EDOM.

## NAME

fclose, fflush - close or flush a stream

## SYNOPSIS

```
#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;
```

## DESCRIPTION

Fclose causes any buffers for the named stream to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling exit(2).

Fflush causes any buffered data for the named output stream to be written to that file. The stream remains open.

## SEE ALSO

close(2), fopen(3), setbuf(3)

## DIAGNOSTICS

These routines return EOF if stream is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

feof, ferror, clearerr, fileno - stream status inquiries

SYNOPSIS

```
#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream;
```

DESCRIPTION

Feof returns non-zero when end of file is read on the named input stream, otherwise zero.

Ferror returns non-zero when an error has occurred reading or writing the named stream, otherwise zero. Unless cleared by clearerr, the error indication lasts until the stream is closed.

Clrerr resets the error indication on the named stream.

Fileno returns the integer file descriptor associated with the stream, see open(2).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3), open(2)

## NAME

fabs, floor, ceil - absolute value, floor, ceiling functions

## SYNOPSIS

```
#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double(x);
```

## DESCRIPTION

Fabs returns the absolute value |x|.

Floor returns the largest integer not greater than x.

Ceil returns the smallest integer not less than x.

## SEE ALSO

abs(3)

## NAME

fopen, freopen, fdopen - open a stream

## SYNOPSIS

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

## DESCRIPTION

Fopen opens the file named by filename and associates a stream with it. Fopen returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r" open for reading

"w" create for writing

"a" append: open for writing at end of file, or create for writing

Freopen substitutes the named file in place of the open stream. It returns the original value of stream. The original stream is closed.

Freopen is typically used to attach the preopened constant names, **stdin**, **stdout**, **stderr**, to specified files.

Fdopen associates a stream with a file descriptor obtained from open, dup, creat, or pipe(2). The type of the stream must agree with the mode of the open file.

## SEE ALSO

open(2), fclose(3)

## DIAGNOSTICS

Fopen and freopen return the pointer **NULL** if filename cannot be accessed.

## BUGS

Fdopen is not portable to systems other than UNIX.

## NAME

fread, fwrite - buffered binary input/output

## SYNOPSIS

```
#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

## DESCRIPTION

Fread reads, into a block beginning at ptr, nitems of data of the type of *ptr from the named input stream. It returns the number of items actually read.

Fwrite appends at most nitems of data of the type of *ptr beginning at ptr to the named output stream. It returns the number of items actually written.

## SEE ALSO

read(2), write(2), fopen(3), getc(3), putc(3), gets(3), puts(3), printf(3), scanf(3)

## DIAGNOSTICS

Fread and fwrite return Ø upon end of file or error.

NAME

frexp, ldexp, modf - split into mantissa and exponent

SYNOPSIS

```
double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;

double modf(value, iptr)
double value, *iptr;
```

DESCRIPTION

Frexp returns the mantissa of a double value as a double quantity, x, of magnitude less than 1 and stores an integer n such that value = x*2**n indirectly through eptr.

Ldexp returns the quantity value*2**exp.

Modf returns the positive fractional part of value and stores the integer part indirectly through iptr.

## NAME

fseek, ftell, rewind - reposition a stream

## SYNOPSIS

```
#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)
```

## DESCRIPTION

Fseek sets the position of the next input or output operation on the stream. The new position is at the signed distance offset bytes from the beginning, the current position, or the end of the file, according as ptrname has the value Ø, 1, or 2.

Fseek undoes any effects of ungetc(3).

Ftell returns the current value of the offset relative to the beginning of the file associated with the named stream. It is measured in bytes on UNIX; on some other systems it is a magic cookie, and the only foolproof way to obtain an offset for fseek.

Rewind(stream) is equivalent to fseek(stream, ØL, Ø).

## SEE ALSO

lseek(2), fopen(3)

## DIAGNOSTICS

Fseek returns -1 for improper seeks.

## NAME

getc, getchar, fgetc, getw - get character or word from stream

## SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;
```

## DESCRIPTION

Getc returns the next character from the named input stream.

Getchar() is identical to getc(stdin).

Fgetc behaves like getc, but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next word from the named input stream. It returns the constant EOF upon end of file or error, but since that is a good integer value, feof and ferror(3) should be used to check the success of getw. Getw assumes no special alignment in the file.

## SEE ALSO

fopen(3), putc(3), gets(3), scanf(3), fread(3), ungetc(3)

## DIAGNOSTICS

These functions return the integer constant EOF at end of file or upon read error.

A stop with message, `Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by fopen.

## BUGS

The end-of-file return from getchar is incompatible with that in UNIX editions 1-6.

Because it is implemented as a macro, getc treats a stream argument with side effects incorrectly. In particular, `getc(*f++);' doesn't work sensibly.

## NAME

getenv - value for environment name

## SYNOPSIS

```
char *getenv(name)
char *name;
```

## DESCRIPTION

Getenv searches the environment list (see environ(5)) for a string of the form name=value and returns value if such a string is present, otherwise Ø (NULL).

## SEE ALSO

environ(5), exec(2)

## NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent - get group file entry

## SYNOPSIS

```
#include <grp.h>

struct group *getgrent();

struct group *getgrgid(gid) int gid;

struct group *getgrnam(name) char *name;

int setgrent();

int endgrent();
```

## DESCRIPTION

Getgrent, getgrgid and getgrnam each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
struct    group { /* see getgrent(3) */
     char *gr_name;
     char *gr_passwd;
     int  gr_gid;
     char **gr_mem;
};
```

The members of this structure are:

gr_name
: The name of the group.

gr_passwd
: The encrypted password of the group.

gr_gid
: The numerical group-ID.

gr_mem
: Null-terminated vector of pointers to the individual member names.

Getgrent simply reads the next line while getgrgid and getgrnam search until a matching gid or name is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to setgrent has the effect of rewinding the group file to allow repeated searches. Endgrent may be called to close the group file when processing is complete.

## FILES

/etc/group

**SEE ALSO**

getlogin(3), getpwent(3), group(5)

**DIAGNOSTICS**

A null pointer (Ø) is returned on EOF or error.

**BUGS**

All information is contained in a static area so it must be copied if it is to be saved.

## NAME

getlogin - get login name

## SYNOPSIS

```
char *getlogin();
```

## DESCRIPTION

Getlogin returns a pointer to the login name as found in /etc/utmp. It may be used in conjunction with getpwnam to locate the correct password file entry when the same userid is shared by several login names.

If getlogin is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to first call getlogin and if it fails, to call getpwuid.

## FILES

/etc/utmp

## SEE ALSO

getpwent(3), getgrent(3), utmp(5)

## DIAGNOSTICS

Returns NULL (Ø) if name not found.

## BUGS

The return values point to static data whose content is overwritten by each call.

## NAME

getpass - read a password

## SYNOPSIS

```
char *getpass(prompt)
char *prompt;
```

## DESCRIPTION

Getpass reads a password from the file /dev/tty, or if that cannot be opened, from the standard input, after prompting with the null-terminated string prompt and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

## FILES

/dev/tty

## SEE ALSO

crypt(3)

## BUGS

The return value points to static data whose content is overwritten by each call.

## NAME

getpw - get name from UID

## SYNOPSIS

```
getpw(uid, buf)
char *buf;
```

## DESCRIPTION

Getpw searches the password file for the (numerical) uid, and fills in buf with the corresponding line; it returns non-zero if uid could not be found. The line is null-terminated.

## FILES

/etc/passwd

## SEE ALSO

getpwent(3), passwd(5)

## DIAGNOSTICS

Non-zero return on error.

## NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent - get password file entry

## SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent();

struct passwd *getpwuid(uid) int uid;

struct passwd *getpwnam(name) char *name;

int setpwent();

int endpwent();
```

## DESCRIPTION

Getpwent, getpwuid and getpwnam each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct    passwd { /* see getpwent(3) */
        char *pw_name;
        char *pw_passwd;
        int  pw_uid;
        int  pw_gid;
        int  pw_quota;
        char *pw_comment;
        char *pw_gecos;
        char *pw_dir;
        char *pw_shell;
};
```

The fields pw_quota and pw_comment are unused; the others have meanings described in passwd(5).

Getpwent reads the next line (opening the file if necessary); setpwent rewinds the file; endpwent closes it.

Getpwuid and getpwnam search from the beginning until a matching uid or name is found (or until EOF is encountered).

## FILES

/etc/passwd

## SEE ALSO

getlogin(3), getgrent(3), passwd(5)

## DIAGNOSTICS

Null pointer (Ø) returned on EOF or error.

## BUGS

All information is contained in a static area so it must be copied if it is to be saved.

## NAME

gets, fgets - get a string from a stream

## SYNOPSIS

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;
```

## DESCRIPTION

Gets reads a string into s from the standard input stream **stdin**. The string is terminated by a newline character, which is replaced in s by a null character. Gets returns its argument.

Fgets reads n-1 characters, or up to a newline character, whichever comes first, from the stream into the string s. The last character read into s is followed by a null character. Fgets returns its first argument.

## SEE ALSO

puts(3), getc(3), scanf(3), fread(3), ferror(3)

## DIAGNOSTICS

Gets and fgets return the constant pointer **NULL** upon end of file or error.

## BUGS

Gets deletes a newline, fgets keeps it, all in the name of backward compatibility.

## NAME

hypot, cabs - euclidean distance

## SYNOPSIS

```
#include <math.h>

double hypot(x, y)
double x, y;

double cabs(z)
struct { double x, y;} z;
```

## DESCRIPTION

Hypot and cabs return

sqrt(x*x + y*y),

taking precautions against unwarranted overflows.

## SEE ALSO

exp(3) for sqrt

NAME

j0, j1, jn, y0, y1, yn - bessel functions

SYNOPSIS

```
#include <math.h>

double j0(x)
double x;

double j1(x)
double x;

double jn(n, x);
double x;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause y0, y1, and yn to return a huge negative value and set errno to EDOM.

## NAME

l3tol, ltol3 - convert between 3-byte integers and long integers

## SYNOPSIS

```
l3tol(lp, cp, n)
long *lp;
char *cp;

ltol3(cp, lp, n)
char *cp;
long *lp;
```

## DESCRIPTION

L3tol converts a list of n three-byte integers packed into a character string pointed to by cp into a list of long integers pointed to by lp.

Ltol3 performs the reverse conversion from long integers (lp) to three-byte integers (cp).

These functions are useful for file-system maintenance; disk addresses are three bytes long.

## SEE ALSO

filsys(5)

## NAME

malloc, free, realloc, calloc - main memory allocator

## SYNOPSIS

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;
```

## DESCRIPTION

Malloc and free provide a simple general-purpose memory allocation package. Malloc returns a pointer to a block of at least size bytes beginning on a word boundary.

The argument to free is a pointer to a block previously allocated by malloc; this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by malloc is overrun or if some random number is handed to free.

Malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls sbrk (see break(2)) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

Realloc also works if ptr points to a block freed since the last call of malloc, realloc or calloc; thus sequences of free, malloc and realloc can exploit the search strategy of malloc to do storage compaction.

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for

storage of any type of object.

## DIAGNOSTICS

Malloc, realloc and calloc return a null pointer (Ø) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. Malloc may be recompiled to check the arena very stringently on every transaction; see the source code.

## BUGS

When realloc returns Ø, the block pointed to by ptr may be destroyed.

## NAME

mktemp - make a unique file name

## SYNOPSIS

```
char *mktemp(template)
char *template;
```

## DESCRIPTION

Mktemp replaces template by a unique file name, and returns the address of the template. The template should look like a file name with six trailing X's, which will be replaced with the current process id and a unique letter.

## SEE ALSO

getpid(2)

## NAME

monitor - prepare execution profile

## SYNOPSIS

```
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
short buffer[ ];
```

## DESCRIPTION

An executable program created by `cc -p' automatically includes calls for monitor with default parameters; monitor needn't be called explicitly except to gain fine control over profiling.

Monitor is an interface to profil(2). Lowpc and highpc are the addresses of two functions; buffer is the address of a (user supplied) array of bufsize short integers. Monitor arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of lowpc and the highest is just below highpc. At most nfunc call counts can be kept; only calls of functions compiled with the profiling option -p of cc(1) are recorded. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

```
extern etext();
...
monitor((int)2, etext, buf, bufsize, nfunc);
```

Etext lies just above all the program text, see end(3).

To stop execution monitoring and write the results on the file mon.out, use

```
monitor(0);
```

then prof(1) can be used to examine the results.

## FILES

mon.out

## SEE ALSO

prof(1), profil(2), cc(1)

## NAME

nlist - get entries from name list

## SYNOPSIS

```
#include <nlist.h>
nlist(filename, nl)
char *filename;
struct nlist nl[ ];
```

## DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to Ø. See a.out(5) for the structure declaration.

This subroutine is useful for examining the system name list kept in the file /unix. In this way programs can obtain system addresses that are up to date.

## SEE ALSO

a.out(5)

## DIAGNOSTICS

All type entries are set to Ø if the file cannot be found or if it is not a valid namelist.

## NAME

perror, sys_errlist, sys_nerr - system error messages

## SYNOPSIS

```
perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];
```

## DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string s is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable errno (see intro(2)), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings sys_errlist is provided; errno can be used as an index in this table to get the message string without the newline. Sys_nerr is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

## SEE ALSO

intro(2)

## NAME

plot: openpl et al. - graphics interface

## SYNOPSIS

```
openpl( )

erase( )

label(s) char s[ ];

line(x1, y1, x2, y2)

circle(x, y, r)

arc(x, y, xØ, yØ, x1, y1)

move(x, y)

cont(x, y)

point(x, y)

linemod(s) char s[ ];

space(xØ, yØ, x1, y1)

closepl( )
```

## DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. See plot(5) for a description of their effect. Openpl must be used before any of the others to open the device for writing. Closepl flushes the output.

String arguments to label and linemod are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following ld(1) options:

| | |
|---|---|
| **-lplot** | device-independent graphics stream on standard output for plot(1) filters |
| **-13ØØ** | GSI 3ØØ terminal |
| **-13ØØs** | GSI 3ØØS terminal |
| **-145Ø** | DASI 45Ø terminal |
| **-14Ø14** | Tektronix 4Ø14 terminal |

## SEE ALSO

plot(5), plot(1), graph(1)

## NAME

popen, pclose - initiate I/O to/from a process

## SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;
```

## DESCRIPTION

The arguments to popen are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by popen should be closed by pclose, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

## SEE ALSO

pipe(2), fopen(3), fclose(3), system(3), wait(2)

## DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

Pclose returns -1 if stream is not associated with a 'popened' command.

## BUGS

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, e.g. with fflush, see fclose(3).

## NAME

printf, fprintf, sprintf - formatted output conversion

## SYNOPSIS

```
#include <stdio.h>

printf(format [, arg ] ... )
char *format;

fprintf(stream, format [, arg ] ... )
FILE *stream;
char *format;

sprintf(s, format [, arg ] ... )
char *s, format;
```

## DESCRIPTION

Printf places output on the standard output stream stdout. Fprintf places output on the named output stream. Sprintf places `output' in the string s, followed by the character `\Ø'.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive arg printf.

Each conversion specification is introduced by the character %. Following the %, there may be

- an optional minus sign `-' which specifies left adjustment of the converted value in the indicated field;
- an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
- an optional period `.' which serves to separate the field width from the next digit string;
- an optional digit string specifying a precision which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

- the character **l** specifying that a following **d**, **o**, **x**, or **u** corresponds to a long integer arg. (A capitalized conversion code accomplishes the same thing.)

- a character which indicates the type of conversion to be applied.

A field width or precision may be `*' instead of a digit string. In this case an integer arg supplies the field width or precision.

The conversion characters and their meanings are

**dox** The integer arg is converted to decimal, octal, or hexadecimal notation respectively.

**f** The float or double arg is converted to decimal notation in the style `[-]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly Ø, no digits and no decimal point are printed.

**e** The float or double arg is converted in the style `[-]d.ddde+dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.

**g** The float or double arg is printed in style **d**, in style **f**, or in style **e**, whichever gives full precision in minimum space.

**c** The character arg is printed. Null characters are ignored.

**s** Arg is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is Ø or missing all characters up to a null are printed.

**u** The unsigned integer arg is converted to decimal and printed (the result will be in the range Ø to 65535).

**%** Print a `%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters

generated by printf are printed by putc(3).

**Examples**
To print a date and time in the form `Sunday, July 3, 10:02', where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day,
    hour, min);
```

To print pi to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

**SEE ALSO**
putc(3), scanf(3), ecvt(3)

**BUGS**
Very wide fields (>128 characters) fail.

## NAME

putc, putchar, fputc, putw - put character or word on a stream

## SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;
```

## DESCRIPTION

Putc appends the character c to the named output stream. It returns the character written.

Putchar(c) is defined as putc(c, stdout).

Fputc behaves like putc, but is a genuine function rather than a macro. It may be used to save on object text.

Putw appends word (i.e. **int**) w to the output stream. It returns the word written. Putw neither assumes nor causes special alignment in the file.

Unlike the original version of stdio, the standard stream stdout is always buffered; this default may be changed by setbuf(3). The standard stream stderr is by default unbuffered unconditionally, but use of freopen (see fopen(3)) will cause it to become buffered; setbuf, again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. Fflush (see fclose(3)) may be used to force the block out early. In addition, all output streams associated with pipes or terminals are automatically flushed before requesting input from a pipe or terminal.

## SEE ALSO

fopen(3), fclose(3), getc(3), puts(3), printf(3), fread(3)

## DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, ferror(3) should be used to detect

putw errors.

## BUGS

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular `putc(c, *f++);' doesn't work sensibly.

## NAME

putc, putchar, fputc, putw - put character or word on a stream

## SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;
```

## DESCRIPTION

Putc appends the character c to the named output stream. It returns the character written.

Putchar(c) is defined as putc(c, stdout).

Fputc behaves like putc, but is a genuine function rather than a macro. It may be used to save on object text.

Putw appends word (i.e. **int**) w to the output stream. It returns the word written. Putw neither assumes nor causes special alignment in the file.

The standard stream stdout is normally buffered if and only if the output does not refer to a terminal; this default may be changed by setbuf(3). The standard stream stderr is by default unbuffered unconditionally, but use of freopen (see fopen(3)) will cause it to become buffered; setbuf, again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. Fflush (see fclose(3)) may be used to force the block out early.

## SEE ALSO

fopen(3), fclose(3), getc(3), puts(3), printf(3), fread(3)

## DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, ferror(3) should be used to detect putw errors.

## BUGS

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular `putc(c, *f++);' doesn't work sensibly.

## NAME

puts, fputs - put a string on a stream

## SYNOPSIS

```
#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;
```

## DESCRIPTION

Puts copies the null-terminated string s to the standard output stream stdout and appends a newline character.

Fputs copies the null-terminated string s to the named output stream.

Neither routine copies the terminal null character.

## SEE ALSO

fopen(3), gets(3), putc(3), printf(3), ferror(3)
fread(3) for fwrite

## BUGS

Puts appends a newline, fputs does not, all in the name of backward compatibility.

## NAME

qsort - quicker sort

## SYNOPSIS

```
qsort(base, nel, width, compar)
char *base;
int (*compar)( );
```

## DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than Ø according as the first argument is to be considered less than, equal to, or greater than the second.

## SEE ALSO

sort(1)

## NAME

rand, srand - random number generator

## SYNOPSIS

```
srand(seed)
int seed;

rand( )
```

## DESCRIPTION

Rand uses a multiplicative congruential random number generator with period 28329 to return successive pseudo-random numbers in the range from Ø to 28159-1.

The generator is reinitialized by calling srand with 1 as argument. It can be set to a random starting point by calling srand with whatever you like as argument.

## NAME

scanf, fscanf, sscanf - formatted input conversion

## SYNOPSIS

```
#include <stdio.h>

scanf(format [ , pointer ] . . .  )
char *format;

fscanf(stream, format [ , pointer ] . . .  )
FILE *stream;
char *format;

sscanf(s, format [ , pointer ] . . .  )
char *s, *format;
```

## DESCRIPTION

Scanf reads from the standard input stream stdin. Fscanf reads from the named input stream. Sscanf reads from the character string s. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string format, described below, and a set of pointer arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.

2. An ordinary character (not %) which must match the next character of the input stream.

3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion

characters are legal:

**%** a single `%' is expected in the input at this point; no assignment is done.

**d** a decimal integer is expected; the corresponding argument should be an integer pointer.

**o** an octal integer is expected; the corresponding argument should be a integer pointer.

**x** a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

**s** a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating `\Ø', which will be added. The input field is terminated by a space character or a newline.

**c** a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try `%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

**e9f7** a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.

**[** indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized or preceeded by **l** to indicate that a pointer to **long** rather

than to **int** is in the argument list. Similarly, the conversion characters **e** or **f** may be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float**. The conversion characters **d**, **o** and **x** may be preceeded by **h** to indicate a pointer to **short** rather than to **int**.

The scanf functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant **EOF** is returned upon end of input; note that this is different from Ø, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; float x; char name[5Ø];
scanf( "%d%f%s", &i, &x, name);
```

with the input line

```
25   54.32E-1  thompson
```

will assign to i the value 25, x the value 5.432, and name will contain `thompson\Ø'. Or,

```
int i; float x; char name[5Ø];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 Ø123 56a72
```

will assign 56 to i, 789.Ø to x, skip `Ø123', and place the string `56\Ø' in name. The next call to getchar will return `a'.

**SEE ALSO**

atof(3), getc(3), printf(3)

**DIAGNOSTICS**

The scanf functions return **EOF** on end of input, and a short count for missing or illegal data items.

**BUGS**

The success of literal matches and suppressed assignments is not directly determinable.

## NAME

setbuf - assign buffering to a stream

## SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;
```

## DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array buf to be used instead of an automatically allocated buffer. If buf is the constant pointer **NULL**, input/output will be completely unbuffered.

A manifest constant **BUFSIZ** tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from malloc(3) upon the first getc or putc(3) on the file, except that output streams directed to terminals, and the standard error stream stderr are normally not buffered.

## SEE ALSO

fopen(3), getc(3), putc(3), malloc(3)

## NAME

setjmp, longjmp - non-local goto

## SYNOPSIS

```
#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;
```

## DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in env for later use by longjmp. It returns value Ø.

Longjmp restores the environment saved by the last call of setjmp. It then returns in such a way that execution continues as if the call of setjmp had just returned the value val to the function that invoked setjmp, which must not itself have returned in the interim. All accessible data have values as of the time longjmp was called.

## SEE ALSO

signal(2)

## NAME

sin, cos, tan, asin, acos, atan, atan2 - trigonometric functions

## SYNOPSIS

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(x, y)
double x, y;
```

## DESCRIPTION

Sin, cos and tan return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range -J/2 to J/2.

Acos returns the arc cosine in the range Ø to J.

Atan returns the arc tangent of x in the range -J/2 to J/2.

Atan2 returns the arc tangent of x/y in the range -J to J.

## DIAGNOSTICS

Arguments of magnitude greater than 1 cause asin and acos to return value Ø; errno is set to EDOM. The value of tan at its singular points is a huge number, and errno is set to ERANGE.

## BUGS

The value of tan for arguments greater than about 2**31 is garbage.

## NAME

sinh, cosh, tanh - hyperbolic functions

## SYNOPSIS

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;
```

## DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

## DIAGNOSTICS

Sinh and cosh return a huge value of appropriate sign when the correct value would overflow.

NAME

sleep - suspend execution for interval

SYNOPSIS

```
sleep(seconds)
unsigned seconds;
```

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

alarm(2), pause(2)

## NAME

stdio - standard buffered input/output package

## SYNOPSIS

```
#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;
```

## DESCRIPTION

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros getc and putc(3) handle characters quickly. The higher level routines gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite all use getc and putc; they can be freely intermixed.

A file with associated buffering is called a stream, and is declared to be a pointer to a defined type **FILE**. Fopen(3) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

| | |
|---|---|
| **stdin** | standard input file |
| **stdout** | standard output file |
| **stderr** | standard error file |

A constant `pointer' **NULL** (Ø) designates no stream at all.

An integer constant **EOF** (-1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following `functions' are implemented as macros; redeclaration of these names is perilous: getc, getchar, putc, putchar, feof, ferror, fileno.

## SEE ALSO

open(2), close(2), read(2), write(2)

## DIAGNOSTICS

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with fopen, input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE**

data.

## NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex - string operations

## SYNOPSIS

```
char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s;
```

## DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string s2 to the end of string s1. Strncat copies at most n characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as s1 is lexicographically greater than, equal to, or less than s2. Strncmp makes the same comparison but looks at at most n characters.

Strcpy copies string s2 to s1, stopping after the null character has been moved. Strncpy copies exactly n characters, truncating or null-padding s2; the target may not be null-terminated if the length of s2 is n or more. Both return s1.

Strlen returns the number of non-null characters in s.

Index (rindex) returns a pointer to the first (last) occurrence of character c in string s, or zero if c does not occur in the string.

BUGS

NAME

swab - swap bytes

SYNOPSIS

```
swab(from, to, nbytes)
char *from, *to;
```

DESCRIPTION

Swab copies nbytes bytes pointed to by from to the position pointed to by to, exchanging adjacent even and odd bytes. It is useful for carrying binary data between different machine types. Nbytes should be even.

## NAME

system - issue a shell command

## SYNOPSIS

```
system(string)
char *string;
```

## DESCRIPTION

System causes the string to be given to sh(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

## SEE ALSO

popen(3), exec(2), wait(2)

## DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

## NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs - terminal independent operation routines

## SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

## DESCRIPTION

These functions extract and use capabilities from the terminal capability data base termcap(5). These are low level routines; see curses(3) for a higher level package.

Tgetent extracts the entry for terminal name into the buffer at bp. Bp should be a character buffer of size 1Ø24 and must be retained through all subsequent calls to tgetnum, tgetflag, and tgetstr. Tgetent returns -1 if it cannot open the termcap file, Ø if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type name is the same as the environment string TERM, the TERMCAP string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than /etc/termcap. This can speed up entry into programs that call tgetent, as well as to help debug new terminal descriptions or to make one for your terminal if you can't write the file /etc/termcap.

Tgetnum gets the numeric value of capability id, returning -1 if is not given for the terminal. Tgetflag returns 1 if the specified capability is present in the terminal's entry, Ø if it is not. Tgetstr gets the string value of capability id, placing it in the buffer at area, advancing the area pointer. It decodes the abbreviations for this field described in termcap(5), except for cursor addressing and padding information.

Tgoto returns a cursor addressing string decoded from cm to go to column destcol in line destline. It uses the external variables UP (from the up capability) and BC (if bc is given rather than bs) if necessary to avoid placing \n, ^D or ^@ in the returned string. (Programs which call tgoto should be sure to turn off the XTABS bit(s), since tgoto may now output a tab. Note that programs using termcap should in general turn off XTABS anyway since some terminals use control I for other functions, such as nondestructive space.) If a % sequence is given which is not understood, then tgoto returns OOPS.

Tputs decodes the leading padding information of the string cp; affcnt gives the number of lines affected by the operation, or 1 if this is not applicable, outc is a routine which is called with each character in turn. The external variable ospeed should contain the output speed of the terminal as encoded by stty (2). The external variable PC should contain a pad character to be used (from the pc capability) if a null (^@) is inappropriate.

**FILES**

| | |
|---|---|
| /usr/lib/libtermcap.a | -ltermcap library |
| /etc/termcap | data base |

**SEE ALSO**

ex(1), curses(3), termcap(5)

**AUTHOR**

William Joy

**BUGS**

## NAME

ttyname, isatty, ttyslot - find name of a terminal

## SYNOPSIS

```
char *ttyname(fildes)

isatty(fildes)

ttyslot()
```

## DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor fildes.

Isatty returns 1 if fildes is associated with a terminal device, Ø otherwise.

Ttyslot returns the number of the entry in the ttys(5) file for the control terminal of the current process.

## FILES

/dev/*
/etc/ttys

## SEE ALSO

ioctl(2), ttys(5)

## DIAGNOSTICS

Ttyname returns a null pointer (Ø) if fildes does not describe a terminal device in directory `/dev'.

Ttyslot returns Ø if `/etc/ttys' is inaccessible or if it cannot determine the control terminal.

## BUGS

The return value points to static data whose content is overwritten by each call.

## NAME

ungetc - push character back into input stream

## SYNOPSIS

```
#include <stdio.h>

ungetc(c, stream)
FILE *stream;
```

## DESCRIPTION

Ungetc pushes the character c back on an input stream. That character will be returned by the next getc call on that stream. Ungetc returns c.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3) erases all memory of pushed back characters.

## SEE ALSO

getc(3), setbuf(3), fseek(3)

## DIAGNOSTICS

Ungetc returns EOF if it can't push a character back.

## NAME

dk - MultiBus disk interface

## DESCRIPTION

Devices with minor device numbers Ø through 7 refer to various partitions of drive Ø. The standard device names begin with "dk" followed by the drive number and then a letter a-h for partitions Ø-7 respectively.

THe block devices access the the disk via the normal system buffering mechanism and may be read and written without regard for physical disk records. There is also a raw interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call generally results in exactly one io operation and is therefore considerably more efficient when many words are transmitted. The names of the raw devices begin with an extra 'r'.

In raw io, counts should be multiples of 1Ø24 bytes (a disk sector). Likewise, seek calls should specify a multiple of 1Ø24 bytes.

## DISK SUPPORT

The driver handles both the Fujitsu Eagle and the Fujitsu Micro disks. The origin and size of all the logical disks on the drive are as follows:

Fujitsu Eagle (5ØØ sectors/cyl = 7DØØØ bytes/cyl)

| device name | cylinders (inclusive) | size (sectors) | length (bytes) | offset (bytes) |
|---|---|---|---|---|
| (none) | Ø | 23 | 5CØØ | Ø |
| rdkØa | 1 - 16 | 7942 | 7C18ØØ | 7DØØØ |
| rdkØb | 17 - 83 | 3344Ø | 2ØA8ØØØ | 84DØØØ |
| rdkØc | Ø - 839 | 42ØØØØ | 19A28ØØØ | Ø |
| rdkØd | 376 - 391 | 7942 | 7C18ØØ | B798ØØØ |
| rdkØe | 392 - 447 | 27968 | 1B5ØØØØ | BF68ØØØ |
| rdkØf | 448 - 839 | 196ØØØ | BF68ØØØ | DACØØØØ |
| rdkØg | 376 - 839 | 232ØØØ | E29ØØØØ | B798ØØØ |
| rdkØh | 84 - 375 | 145673 | 8E424ØØ | 29Ø4ØØØ |

| | | | | |
|---|---|---|---|---|
| (none) | 840 - 842 | 1500 | 177000 | 19A28000 |

Fujitsu Microdisk (126 sectors/cyl = 1F800 bytes/cyl)

| device name | cylinders (inclusive) | size (sectors) | length (bytes) | offset (bytes) |
|---|---|---|---|---|
| (none) | 0 | 23 | 5C00 | 0 |
| rdk0a | 1 - 64 | 7942 | 7C1800 | 1F800 |
| rdk0b | 65 - 197 | 16720 | 1054000 | 7FF800 |
| rdk0c | 0 - 586 | 73962 | 483A800 | 0 |
| rdk0d | 198 - 261 | 7942 | 7C1800 | 185D000 |
| rdk0e | 262 - 483 | 27968 | 1B50000 | 203D000 |
| rdk0f | 484 - 586 | 12978 | CAC800 | 3B8E000 |
| rdk0g | 262 - 586 | 40950 | 27FD800 | 203D000 |
| (none) | 587 - 589 | 378 | 5E800 | 483A800 |

## FILES

| | |
|---|---|
| /dev/dk0[a-h] | block devices |
| /dev/rdk0[a-h] | raw devices |

## BUGS

In raw io, byte counts and file offsets are rounded up to 1024-byte block boundaries. If counts which are not multiples of 1024-bytes are used, unwanted io will result. Thus, any programs accessing the raw devices using read, write, and lseek(2) should always deal in 1024-byte multiples.

## NAME

mem, kmem - core memory

## DESCRIPTION

Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system. Kmem is the same as mem except that kernel virtual memory rather than physical memory is accessed.

Byte addresses are interpreted as memory addresses. References to non-existent locations return errors.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

On PDP11's, the I/O page begins at location 0160000 of kmem and per-process data for the current process begins at 0140000.

## FILES

/dev/mem, /dev/kmem

## NAME

mouse - optical mouse driver

## DESCRIPTION

The mouse driver provides an interface to an optical mouse. The mouse data consists of an x-position, y-position, and switch states. The mouse driver maintains the mouse data and provides a signal (SIGMOUS) when any of the data changes.

The mouse must be opened before use, which enables the signals. The process which opens the mouse is the only process which will get the signals. When the mouse is closed it is available for other pocesses.

The mouse data may be sampled at any time using an ioctl call (WIOCRDMS). The ioctl returns a long: bits Ø-11 contain the 2's compliment of the current x position, bits 16-27 contain the 2's compliment of the current y position, bit 12 is the state of the right switch (Ø=depressed), bit 13 is the state of the middle switch (Ø=depressed), and bit 14 is the state of the left switch (Ø=depressed). All other bits are not used and should be masked out.

The mouse driver is designed for use with a Mouse Systems Corporation M-1 Mouse. The mouse should be set for 12ØØ baud, non-rotatable, self-test disabled, vertical blue lines, normal, and TTL interface (switches 4 and 6 on, all others off).

## FILES

/dev/mouse (character device, major=5, minor=Ø)

```
#include <ioctl.h>
#include <signal.h>
```

## NAME

newtty - a new terminal interface

## SYNOPSIS

**stty new**

## DESCRIPTION

This section describes a new terminal interface, which is available as a replacement for that described in tty(4). It is necessary to use this terminal interface to take advantage of the full process control power of csh(1), as it is not possible to generate the signals necessary to stop processes gracefully from a keyboard controlled by the discipline described in tty(4). This interface provides ``better'' editing functionality over terminal input than does tty(4) extracting its price in added complexity.

To enable the discipline in a program, one uses the TIOCSETD ioctl, e.g.:

```
#include <sgtty.h>

int ldisc = NTTYDISC;

ioctl(0, TIOCSETD, &ldisc);
```

Replacing TIOCSETD with TIOCGETD will yield in ldisc the current discipline, which will be either 0 (if tty(4) is being used), NETDISC if bk(4) is being used, or NTTYDISC.

The file /dev/tty is, in each process, a synonym for the control terminal associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

As for terminals in general: all of the low-speed asynchronous communications ports use, by default, the same general interface, no matter what hardware is involved. That interface is described in tty(4). The interface described in bk(4) is used for networking applications, and whenever high-speed input must be presented to the machine. This newtty interface described here provides the features of tty(4) and also additional features required for the job control mechanisms provided by csh(1). It also provides input editing functions and options not found in tty(4); these functions are reminiscent of those provided on other systems such as TOPS-20.

N.B.: This line discipline is provided in support of the

mechanisms of csh(1). Its default mode is highly compatible with the standard UNIX version 7 tty driver, tty(4), but programs which are written to require features of this driver are less portable than they would be if they could make do without them.

The remainder of this section discusses the common features of the interface; the presentation parallels that of tty(4).

When a terminal file is opened, it causes the process to wait until a connection is established. In practice user's programs seldom open these files; they are opened by init and become a user's input and output file. The very first terminal file open in a process becomes the control terminal for that process, and the system establishes a process group consisting initially of that single process, associating this group with the terminal. The control terminal's process group plays a special role in handling the quit, interrupt, and stop signals, as discussed below. The control terminal is inherited by a child process during a fork, even if the control terminal is closed. The set of processes which are in the process group associated with the terminal receive certain signals together, as described below and also in kill(2).

By moving sets of related processes in and out of the terminals process group with setpgrp(2), shells can allow the user to arbitrate between these different jobs. The terminal driver aids this arbitration by sending a SIGTTIN signal to the process group of a process which attempts to read from its control terminal while it is not in the terminals process group. If the action for the SIGTTIN signal is SIG_IGN or SIG_HOLD, then no signal is sent; instead the process gets an end-of-file on the read. A signal SIGTTOU exists and has a similar function for output. The default, however, is to allow background processes to produce output. They are stopped only if the local tty option bit TOSTOP is set. (If the process chooses to ignore or hold SIGTTOU signals while the TOSTOP bit is set, then the output is simply discarded; it is not clear that this is a reasonable thing to do, but it is hard to know what to do in this situation.)

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently this limit is 256 characters. When the input limit is reached further characters are ignored and the system rings the terminal

bell to warn that no more buffer space is available.

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. There are special modes, discussed below, that permit the program to read each character as typed without waiting for a full line, or to be informed by a SIGTINT interrupt whenever data is available to read.

During input, erase and kill processing is normally done. By default, either a `#' or a backspace (control-H) character erases the last character typed, except that it will not erase beyond the beginning of a line or an EOT. By default, either the character `@' or the character NAK (control-U) kills the entire line up to the point where it was typed, but not beyond an EOT. These characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Each of them may be entered literally by preceding it by `\'; the erase or kill character remains, but the `\' disappears. These characters may be changed to others. The intention in providing two erase and two kill characters is that the characters ^H and ^U should always be available. The default `#' and `@' characters are often eliminated so that these characters can be typed normally, or to provide a different erase or kill character which a particular user likes.

When desired, all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by `\', and upper-case letters are printed preceded by `\' on output. In addition, the following escape sequences can be generated on output and accepted on input:

| for | use |
|---|---|
| ` | \' |
| \| | \! |
| ~ | \^ |
| { | \( |
| } | \) |

Besides ^H and ^U some other ASCII control characters have special meaning. These characters are not passed to a reading program except in raw mode where they lose their special character, or when typed immediately after the literal next character, normally ^V. Also, it is possible to change these characters from the default; see below.

ETB (Control-W) erases the preceding word (but not any spaces before it). A word is a sequence of non-blank characters (tabs count as ``blank'' here.)

EOT (Control-D) may be used to generate an end of file from a terminal. When an EOT is received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOT is discarded. Thus if there are no characters waiting, which is to say the EOT occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file indication.

DEL (Rubout) is not passed to a program but generates an interrupt signal which is sent to all processes in the process group of the control terminal. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. See signal(2).

ETX (Control-C) also generates an interrupt as a synonym for DEL. Some users may make DEL an erase character, but ^C should normally work on all terminals, unless the user (impolitely) changes both interrupt characters.

EM (Control-Z) is not passed to a program but generates a stop signal (SIGTSTOP) which is sent to all processes in the process group of the controlling terminal, normally suspending the current job, so that it may be placed in the background or another job may be run. The job can later be continued or killed using facilities of the shell.

SUB (Control-Y) generates a SIGTSTOP signal as ^Z does, but the signal is sent when a program tries to read the ^Y, rather than when it is typed.

DC2 (Control-R) retypes the pending input on a new line. (Retyping automagically occurs if characters which would normally be erased from the screen are fouled by program output.)

US (Control-O) flushes the output to the terminal until any character is typed, or a program reads from the terminal. Programs (such as the shell) which print prompts often clear the state set by ^O so that their prompt will appear.

SYN (Control-V) is a causes the next character input to be taken literally. It has no special function. Any character may be input in this way, including ^V.

FS (Control-\ or control-shift-L) generates the quit signal. Its treatment is identical to the interrupt signal except that unless a receiving process has made other arrangements it will not only be terminated but a core image file will be generated.

DC3 (Control-S) delays all printing on the terminal until something is typed in.

DC1 (Control-Q) restarts printing after DC3 without generating any input to a program.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a hangup signal is sent to all processes with the terminal as control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file on their input can terminate appropriately when hung up on.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is always generated on output. The EOT character is not transmitted (except in raw or cbreak mode) to prevent terminals that respond to it from hanging up.

Several ioctl(2) calls apply to terminals. The standard calls are described in tty(4) and will not be repeated here. Additional functions provided by this interface use bits in a local mode word, described in <ioctl.h> as follows:

| | | |
|---|---|---|
| LCRTBS | 000001 | Backspace on erase rather than echoing erase |
| LPRTERA | 000002 | Printing terminal erase mode |
| LCRTERA | 000004 | Erase character echoes as backspace-space-backspac |
| LTILDE | 000010 | Convert ~ to ' on output (for Hazeltine terminals) |
| LMDMBUF | 000020 | Stop/start output when carrier drops |
| LLITOUT | 000040 | Suppress output translations |
| LTOSTOP | 000100 | Send SIGTTYO for background output |
| LFLUSHO | 000200 | Output is being flushed |
| LNOHANG | 000400 | Don't send hangup when carrier drops |
| LETXACK | 001000 | Diablo style buffer hacking (unimplemented) |
| LCRTKIL | 002000 | BS-space-BS erase entire line on line kill |
| LINTRUP | 004000 | Generate interrupt SIGTINT when input ready to rea |
| LCTLECH | 010000 | Echo input control chars as ^X, delete as ^? |

LPENDIN 020000 Retype pending input at next read or input charac..

The applicable ioctl functions are:

TIOCLBIS
arg is a (integer) mask which is the bits to be set in the local mode word.

TIOCLBIC
arg is a mask of bits to be cleared.

TIOCLSET
arg is a new mask to be placed in the local mode word.

TIOCLGET
arg is the address of a word which the current mask is placed in.

TIOCSBRK
the break bit is set in the terminal.

TIOCCBRK
the break bit is cleared.

TIOCSDTR
data terminal ready is set.

TIOCCDTR
data terminal ready is cleared.

TIOCGPGRP
arg is the address of a word into which is placed the process group number of the current terminal.

TIOCSPGRP
args is a word (typically a process id) which becomes the process group for the control terminal.

TIOCSLTC
args is the address of a ltchars structure which defines the new local special characters.

TIOCGLTC
args is the address of a ltchars structure into which is placed the current set of local special characters.

FIONREAD
returns in the long integer whose address is arg the number of immediately readable characters from the argument unit. This works for files, pipes, and terminals, but not (yet) for multiplexed channels.

The ltchars structure is defined by:

```
struct ltchars {
        char t_suspc;  /* stop process signal */
        char t_dstopc; /* delayed stop process signal */
        char t_rprntc; /* reprint line */
        char t_flushc; /* flush output (toggles) */
        char t_werasec;      /* word erase */
        char t_lnextc; /* literal next character */
        char t_lerasec;      /* local erase */
        char t_lkill;        /* local kill */
        char t_lintr;  /* local interrupt */
};
```

The default values for these characters are ^Z, ^Y, ^R, ^O, ^W, ^V, ^H, ^U and ^? (delete). A value of -1 disables the character.

**FILES**

/dev/tty
/dev/tty*
/dev/console

**SEE ALSO**

getty(8), stty (1), signal(2), ioctl(2), bk(4), tty(4)

**BUGS**

The output flush character ^O incorrectly flushes input as well.

## NAME

null - data sink

## DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return Ø bytes.

## FILES

/dev/null

## NAME

tm - MultiBus magtape interface

## DESCRIPTION

The files mtØ and mt4 refer to the half-inch Cipher magtape drive. The file mtØ is rewound when it is closed, the file mt4 is not. When a file open for writing is closed, two end-of-file marks are written. If the tape is not to be rewound it is positioned with the head between the two marks.

A standard tape consists of a series of 1Ø24-byte records terminated by an end-of-file. To the extent possible, the tape is treated like any other file (although this is often very inefficient). Although is is possible to read or write one byte at a time, writing in very small units is inadvisable because it tends to create enormous record gaps.

The file mtØ is useful when it is desirable to access the tape in a way compatible with ordinary files. When dealing with foreign tapes, or when reading or writing long records, the "raw" interface is appropriate. The maximum buffer size for the raw interface is 64k-1 bytes. The "raw" interface can be accessed via the files rmtØ and rmt4, where rmtØ is rewound after closing and rmt4 is not. Several ioctl operations are available on raw magnetic tape. The following definitions are from <sys/mtio.h>:

```
/*
 * Structures and definitions for mag tape io control commands
 */

#ifndef _IOCTL_
#include <sys/ioctl.h>
#endif

/* structure for MTIOCTOP - mag tape op command */

struct    mtop {
     short     mt_op;          /* operations defined below */
     short     mt_count; /* how many of them */
};

/* operations: */
#define MTWEOF Ø    /* write an end-of-file record */
#define MTFSF  1    /* forward space file */
#define MTBSF  2    /* backward space file */
#define MTFSR  3    /* forward space record */
#define MTBSR  4    /* backward space record */
#define MTREW  5    /* rewind */
#define MTOFFL 6    /* rewind and put the drive offline */
#define MTNOP   7       /* nop */
```

```
#define MTERASE 8       /* erase to end of tape */

/* structure for MTIOCGET - mag tape get status command */

struct    mtget     {
     short     mt_dsreg; /* drive status register */
     short     mt_erreg; /* error register */
     short     mt_resid; /* residual count */
/* the following two are not yet implemented */
     short     mt_fileno;     /* file number of current position */
     short     mt_blkno; /* block number of current position */
};

/* the bits in these registers are grossly device dependent */
/* should be defined here but aren't */
```

Each read or write call reads or writes the next record on the tape. For writes, the record length is the size of the buffer given. For a read, the number of bytes read is the lesser of the buffer size and the record size. For raw tape io, a seek is ignored.

**FILES**

| | |
|---|---|
| /dev/mt[Ø,4] | block devices |
| /dev/rmt[Ø,4] | raw devices |

**BUGS**

An error should be returned if the record size is greater than the read buffer size.

NAME

tty - general terminal interface

DESCRIPTION

This section describes both a particular special file, and the general nature of the terminal interface.

The file /dev/tty is, in each process, a synonym for the control terminal associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

As for terminals in general: all of the low-speed asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of the interface.

When a terminal file is opened, it causes the process to wait until a connection is established. In practice user's programs seldom open these files; they are opened by init and become a user's input and output file. The very first terminal file open in a process becomes the control terminal for that process. The control terminal plays a special role in handling quit or interrupt signals, as discussed below. The control terminal is inherited by a child process during a fork, even if the control terminal is closed. The set of processes that thus share a control terminal is called a process group; all members of a process group receive certain signals together, see DEL below and kill(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently this limit is 256 characters. When the input limit is reached all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. There are special modes, discussed

below, that permit the program to read each character as typed without waiting for a full line.

During input, erase and kill processing is normally done. By default, the character `#' erases the last character typed, except that it will not erase beyond the beginning of a line or an EOT. By default, the character `@' kills the entire line up to the point where it was typed, but not beyond an EOT. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Either `@' or `#' may be entered literally by preceding it by `\'; the erase or kill character remains, but the `\' disappears. These two characters may be changed to others.

When desired, all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by `\'. In addition, the following escape sequences can be generated on output and accepted on input:

| for | use |
|---|---|
| ` | \' |
| \| | \! |
| ~ | \^ |
| { | \( |
| } | \) |

Certain ASCII control characters have special meaning. These characters are not passed to a reading program except in raw mode where they lose their special character. Also, it is possible to change these characters from the default; see below.

EOT (Control-D) may be used to generate an end of file from a terminal. When an EOT is received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOT is discarded. Thus if there are no characters waiting, which is to say the EOT occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file indication.

DEL (Rubout) is not passed to a program but generates an interrupt signal which is sent to all processes with the associated control terminal. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. See signal(2).

FS (Control-\ or control-shift-L) generates the quit signal. Its treatment is identical to the interrupt

signal except that unless a receiving process has made other arrangements it will not only be terminated but a core image file will be generated.

DC3 (Control-S) delays all printing on the terminal until something is typed in.

DC1 (Control-Q) restarts printing after DC3 without generating any input to a program.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a hangup signal is sent to all processes with the terminal as control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file on their input can terminate appropriately when hung up on.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is always generated on output. The EOT character is not transmitted (except in raw mode) to prevent terminals that respond to it from hanging up.

Several ioctl(2) calls apply to terminals. Most of them use the following structure, defined in <sgtty.h>:

```
struct sgttyb {
     char sg_ispeed;
     char sg_ospeed;
     char sg_erase;
     char sg_kill;
     int  sg_flags;
};
```

The sg_ispeed and sg_ospeed fields describe the input and output speeds of the device according to the following table, which corresponds to the DEC DH-11 interface. If other hardware is used, impossible speed changes are ignored. Symbolic values in the table are as defined in <sgtty.h>.

| | | |
|---|---|---|
| BØ | Ø | (hang up dataphone) |
| B5Ø | 1 | 5Ø baud |

| | | |
|---|---|---|
| B75 | 2 | 75 baud |
| B11Ø | 3 | 11Ø baud |
| B134 | 4 | 134.5 baud |
| B15Ø | 5 | 15Ø baud |
| B2ØØ | 6 | 2ØØ baud |
| B3ØØ | 7 | 3ØØ baud |
| B6ØØ | 8 | 6ØØ baud |
| B12ØØ | 9 | 12ØØ baud |
| B18ØØ | 1Ø | 18ØØ baud |
| B24ØØ | 11 | 24ØØ baud |
| B48ØØ | 12 | 48ØØ baud |
| B96ØØ | 13 | 96ØØ baud |
| EXTA | 14 | External A |
| EXTB | 15 | External B |

In the current configuration, only 11Ø, 15Ø, 3ØØ and 12ØØ baud are really supported on dial-up lines. Code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program. The half-duplex line discipline required for the 2Ø2 dataset (12ØØ baud) is not supplied; full-duplex 212 datasets work fine.

The sg erase and sg kill fields of the argument structure specify the erase and kill characters respectively. (Defaults are # and @.)

The sg flags field of the argument structure contains several bits that determine the system's treatment of the terminal:

| | | |
|---|---|---|
| ALLDELAY | Ø177ØØ | Delay algorithm selection |
| BSDELAY | Ø1ØØØØ | Select backspace delays (not implemented): |
| BSØ | Ø | |
| BS1 | Ø1ØØØØ | |
| VTDELAY | ØØ4ØØØ | Select form-feed and vertical-tab delays: |
| FFØ | Ø | |
| FF1 | Ø1ØØØØ | |
| CRDELAY | ØØ3ØØØ | Select carriage-return delays: |
| CRØ | Ø | |
| CR1 | ØØ1ØØØ | |
| CR2 | ØØ2ØØØ | |
| CR3 | ØØ3ØØØ | |
| TBDELAY | ØØØ6ØØ | Select tab delays: |
| TABØ | Ø | |
| TAB1 | ØØØ1ØØ | |
| TAB2 | ØØØ4ØØ | |
| XTABS | ØØØ6ØØ | |
| NLDELAY | ØØØ14Ø | Select new-line delays: |
| NLØ | Ø | |
| NL1 | ØØØ4ØØ | |
| NL2 | ØØØ1ØØ | |
| NL3 | ØØØ14Ø | |

| | | |
|---|---|---|
| EVENP | 0000200 | Even parity allowed on input (most terminals) |
| ODDP | 0000100 | Odd parity allowed on input |
| RAW | 0000040 | Raw mode: wake up on all characters, 8-bit interface |
| CRMOD | 0000020 | Map CR into LF; echo LF or CR as CR-LF |
| ECHO | 0000010 | Echo (full duplex) |
| LCASE | 0000004 | Map upper case to lower on input |
| CBREAK | 0000002 | Return each character as soon as typed |
| TANDEM | 0000001 | Automatic flow control |

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

Backspace delays are currently ignored but might be used for Terminet 300's.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is unimplemented and is 0.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

Characters with the wrong parity, as determined by bits 200 and 100, are ignored.

In raw mode, every character is passed immediately to the program without waiting until a full line has been typed. No erase or kill processing is done; the end-of-file indicator (EOT), the interrupt character (DEL) and the quit character (FS) are not treated specially. There are no delays and no echoing, and no replacement of one character for another; characters are a full 8 bits for both input and output (parity is up to the program).

Mode 020 causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (for terminals with a new-line function).

CBREAK is a sort of half-cooked (rare?) mode. Programs can

read each character as soon as typed, instead of waiting for a full line, but quit and interrupt work, and output delays, case-translation, CRMOD, XTABS, ECHO, and parity work normally. On the other hand there is no erase or kill, and no special treatment of \ or EOT.

TANDEM mode causes the system to produce a stop character (default DC3) whenever the input queue is in danger of overflowing, and a start character (default DC1) when the input queue has drained sufficiently. It is useful for flow control when the `terminal' is actually another machine that obeys the conventions.

Several ioctl calls have the form:

```
#include <sgtty.h>

ioctl(fildes, code, arg)
struct sgttyb *arg;
```

The applicable codes are:

TIOCGETP
: Fetch the parameters associated with the terminal, and store in the pointed-to structure.

TIOCSETP
: Set the parameters according to the pointed-to structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOCSETN
: Set the parameters but do not delay or flush input. Switching out of RAW or CBREAK mode may cause some garbage input.

With the following codes the arg is ignored.

TIOCEXCL
: Set ``exclusive-use'' mode: no further opens are permitted until the file has been closed.

TIOCNXCL
: Turn off ``exclusive-use'' mode.

TIOCHPCL
: When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with an ACU used to place outgoing calls.

TIOCFLUSH

All characters waiting in input or output queues are flushed.

The following codes affect characters that are special to the terminal interface. The argument is a pointer to the following structure, defined in <sgtty.h>:

```
struct tchars {
        char t_intrc;       /* interrupt */
        char t_quitc;       /* quit */
        char t_startc;  /* start output */
        char t_stopc;   /* stop output */
        char t_eofc;        /* end-of-file */
        char t_brkc;        /* input delimiter (like nl) */
};
```

The default values for these characters are DEL, FS, DC1, DC3, EOT, and -1. A character value of -1 eliminates the effect of that character. The t_brkc character, by default -1, acts like a new-line in that it terminates a `line,' is echoed, and is passed to the program. The `stop' and `start' characters may be the same, to produce a toggle effect. It is probably counterproductive to make other special characters (including erase an kill) identical.

The calls are:

TIOCSETC
Change the various special characters to those given in the structure.

TIOCSETP
Set the special characters to those given in the structure.

**FILES**

/dev/tty
/dev/tty*
/dev/console

**SEE ALSO**

getty(8), stty (1), signal(2), ioctl(2)

**BUGS**

Half-duplex terminals are not supported.

The terminal handler has clearly entered the race for ever-greater complexity and generality. It's still not complex and general enough for TENEX fans.

## NAME

a.out - assembler and link editor output

## SYNOPSIS

```
#include <a.out.h>
```

## DESCRIPTION

A.out is the output file of the assembler as(1) and the link editor ld(1). Both programs make a.out executable if there were no errors and no unresolved external references. Layout information as given in the include file is:

```
/*
 * Header prepended to each a.out file.
 */
struct exec {
        long    a_magic;    /* magic number */
        long    a_text;     /* size of text segment */
        long    a_data;     /* size of initialized data */
        long    a_bss;      /* size of uninitialized data */
        long    a_syms;     /* size of symbol table */
        long    a_trsize;   /* size of text relocation */
        long    a_drsize;   /* size of data relocation */
        long    a_entry;    /* entry point */
};

#define OMAGIC  0407        /* old impure format */
#define NMAGIC  0410        /* read-only text */
#define ZMAGIC  0413        /* demand load format */

/*
 * Macros which take exec structures as arguments and tell whether
 * the file has a reasonable magic number or offsets to text|symbols|s
 */
#define N_BADMAG(x)     (((x).a_magic)!=OMAGIC && ((x).a_magic)!=NMAGI

#define N_TXTOFF(x)         ((x).a_magic==ZMAGIC ? 1024 : sizeof (stru
#define N_SYMOFF(x)         (N_TXTOFF(x) + (x).a_text+(x).a_data + (x)
#define N_STROFF(x)         (N_SYMOFF(x) + (x).a_syms)

/*
 * Format of a relocation datum.
 */
struct relocation_info {
        long    r_address; /* address which is relocated */
unsigned short  r_symbolnum,/* local symbol ordinal */
                r_pcrel:1, /* was relocated pc relative already */
                r_length:2,/* 0=byte, 1=word, 2=long */
                r_extern:1,/* does not include value of sym reference
                :4;        /* nothing, yet */
};
```

```
/*
 * Format of a symbol table entry; this file is included by <a.out.h>
 * and should be used if you aren't interested the a.out header
 * or relocation information.
 */
struct  nlist {
        union {
                char    *n_name;/* for use when in-core */
                long    n_strx;/* index into file string table */
        } n_un;
        unsigned char n_type;/* type flag, i.e. N_TEXT etc; see below
        char    n_other;    /* unused */
        short   n_desc;     /* see <stab.h> */
# ifndef pdp11
        unsigned long n_value;/* value of this symbol (or sdb offset)
# else
        long n_value;
# endif
};
#define n_hash  n_desc      /* used internally by ld */

/*
 * Simple values for n_type.
 */
#define N_UNDF  0x0         /* undefined */
#define N_ABS   0x2         /* absolute */
#define N_TEXT  0x4         /* text */
#define N_DATA  0x6         /* data */
#define N_BSS   0x8         /* bss */
#define N_COMM  0x12        /* common (internal to ld) */
#define N_FN    0x1e        /* file name symbol */

#define N_EXT   01          /* external bit, or'ed in */
#define N_TYPE  0x1e        /* mask for all the type bits */

/*
 * Sdb entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */
#define N_STAB  0xe0        /* if any of these bits set, a SDB entry *

/*
 * Format for namelist values.
 */
#define N_FORMAT "%08lx"
```

The file has four sections: a header, the program and data text, relocation information, and a symbol table (in that order). The last two may be empty if the program was loaded with the `-s' option of ld or if the symbols and relocation have been removed by strip(1).

In the header the sizes of each section are given in bytes, but are even. The size of the header is not included in any of the other sizes.

When an a.out file is loaded into core for execution, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all Ø, following initialized), and a stack. The text segment begins at Ø in the core image; the header is not loaded. If the magic number in the header is Ø4Ø7(8), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. If the magic number is Ø41Ø, the data segment begins at the first Ø mod 8K byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. If the magic number is 411, the text segment is again pure, write-protected, and shared, and moreover instruction and data space are separated; the text and data segment both begin at location Ø. If the magic number is Ø4Ø5, the text segment is overlaid on an existing (Ø411 or Ø4Ø5) text segment and the existing data segment is preserved.

The stack will occupy the highest possible locations in the core image: from Ø177776(8) and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by brk(2).

The start of the text segment in the file is Ø2Ø(8); the start of the data segment is Ø2Ø+S9t8 (the size of the text) the start of the relocation information is Ø2Ø+S9t8+S9d8; the start of the symbol table is Ø2Ø+2(S9t8+S9d8) if the relocation information is present, Ø2Ø+S9t8+S9d8 if not.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file. Other flag values may occur if an assembly language program defines machine instructions.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader ld as the name of a common region whose size is indicated by the value of the symbol.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in core when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information for that word, then the value of the word as stored in the file is an offset from the

associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the `relocation info stripped' flag in the header is on.

Bits 3-1 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

```
Ø0Ø  absolute number
Ø02  reference to text segment
Ø04  reference to initialized data
Ø06  reference to uninitialized data (bss)
Ø1Ø  reference to undefined external symbol
```

Bit Ø of the relocation word indicates, if 1, that the reference is relative to the pc (e.g. `clr x'); if Ø, that the reference is to the actual symbol (e.g., `clr *$x').

The remainder of the relocation word (bits 15-4) contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered Ø, the second 1, etc.

**SEE ALSO**

as(1), ld(1), nm(1)

## NAME

acct - execution accounting file

## SYNOPSIS

```
#include <sys/acct.h>
```

## DESCRIPTION

Acct(2) causes entries to be made into an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

```
/*
        (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983.  ALL
        RIGHTS RESERVED.  PROPERTY OF TEXAS INSTRUMENTS INCORPORATED.
        RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE IS SUBJECT
        TO RESTRICTIONS SET FORTH IN TI'S PROGRAM LICENSE AGREEMENT AN
        ASSOCIATED DOCUMENTATION.
*/

#ifdef SCCSID
static char id_acct[] = "@(#)acct.h1.2 (Texas Instruments) 83/06/27";
#endif

#ifndef SCCSID

/*
 * Accounting structures
 */

typedef unsigned short comp_t;/* "floating pt": 3 bits base 8 exp, 13
struct  acct
{
        char    ac_comm[10]; /* Accounting command name */
        comp_t  ac_utime;    /* Accounting user time */
        comp_t  ac_stime;    /* Accounting system time */
        comp_t  ac_etime;    /* Accounting elapsed time */
        time_t  ac_btime;    /* Beginning time */
        short   ac_uid;      /* Accounting user ID */
        short   ac_gid;      /* Accounting group ID */
        short   ac_mem;      /* average memory usage */
        comp_t  ac_io;       /* number of disk IO blocks */
        dev_t   ac_tty;      /* control typewriter */
        char    ac_flag;     /* Accounting flag */
};

extern  struct  acct        acctbuf;
extern  struct  inode       *acctp;/* inode of accounting file */

#define AFORK   01          /* has executed fork, but no exec */
#define ASU     02          /* used super-user privileges */
```

```
#endif
```

If the process does an exec(2), the first 10 characters of the filename appear in ac comm. The accounting flag contains bits indicating whether exec(2) was ever accomplished, and whether the process ever had super-user privileges.

**SEE ALSO**

acct(2), sa(1)

## NAME

ar - archive (library) file format

## SYNOPSIS

```
#include <ar.h>
```

## DESCRIPTION

The archive command ar is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor ld.

A file produced by ar has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG "!<arch>Ø
#define SARMAG8

#define ARFMAG"`Ø

struct ar_hdr {
        char  ar_name[16];
        char  ar_date[12];
        char  ar_uid[6];
        char  ar_gid[6];
        char  ar_mode[8];
        char  ar_size[1Ø];
        char  ar_fmag[2];
};
```

The name is a null-terminated string; the date is in the form of time(2); the user ID and group ID are numbers; the mode is a bit pattern per chmod(2); the size is counted in bytes.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

## SEE ALSO

ar(1), ld(1), nm(1)

## BUGS

Coding user and group IDs as characters is a botch.

## NAME

core - format of core image file

## DESCRIPTION

UNIX writes out a core image of a terminated process when any of various errors occur. See signal(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called `core' and is written in the process's working directory (provided it can be; normal access controls apply).

The first 1Ø24 bytes of the core image are a copy of the system's per-user data for the process, including the registers as they were at the time of the fault; see the system listings for the format of this area. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is write-protected and shared, it is not dumped; otherwise the entire address space is dumped.

In general the debugger adb(1) is sufficient to deal with core images.

## SEE ALSO

adb(1), signal(2)

**NAME**

dir - format of directories

**SYNOPSIS**

```
#include <sys/dir.h>
```

**DESCRIPTION**

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry see, filsys(5). The structure of a directory entry as given in the include file is:

```
/*
        (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983. ALL
        RIGHTS RESERVED. PROPERTY OF TEXAS INSTRUMENTS INCORPORATED.
        RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE IS SUBJECT
        TO RESTRICTIONS SET FORTH IN TI'S PROGRAM LICENSE AGREEMENT AND
        ASSOCIATED DOCUMENTATION.
*/

#ifdef SCCSID
static char id_dir[] = "@(#)dir.h1.2 (Texas Instruments) 83/Ø6/27";
#endif

#ifndef SCCSID

#ifndef DIRSIZ
#define DIRSIZ14
#endif
struct  direct
{
        ino_t d_ino;
        char  d_name[DIRSIZ];
};

#endif
```

By convention, the first two entries in each directory are for `.' and `..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of `..' is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. Therefore in both cases `..' has the same meaning as `.'.

**SEE ALSO**

filsys(5)

## NAME

dump, ddate - incremental dump format

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ino.h>
# include <dumprestor.h>
```

## DESCRIPTION

Tapes used by dump and restor(1) contain:

a header record
two groups of bit map records
a group of records describing directories
a group of records describing files

The format of the header record and of the first record of each description as given in the include file <dumprestor.h> is:

```
#define NTREC     20/* this may need to change to 1 */
#define MLEN      16
#define MSIZ      4096

#define TS_TAPE   1
#define TS_INODE  2
#define TS_BITS   3
#define TS_ADDR   4
#define TS_END    5
#define TS_CLRI   6
#define MAGIC     (int)60011
#define CHECKSUM  (int)84446
struct            spcl
{
     int          c_checksum;/* this needs to be int aligned */
     int          c_type;
     int          c_magic;
     int          c_volume;
     int          c_count;
     time_t       c_date;
     time_t       c_ddate;
     daddr_t      c_tapea;
     ino_t        c_inumber;
     struct       dinodec_dinode;
     char         c_addr[BSIZE];
} spcl;

struct            idates
{
     char         id_name[16];
     char         id_incno;
     time_t       id_ddate;
```

```
};
```

NTREC is the number of 512 byte records in a physical tape block. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the c_type field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE Tape volume label

TS_INODE
: A file or directory follows. The c_dinode field is a copy of the disk inode and contains bits telling what sort of file this is.

TS_BITS A bit map follows. This bit map has a one bit for each inode that was dumped.

TS_ADDR A subrecord of a file description. See c_addr below.

TS_END End of tape record.

TS_CLRI A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.

MAGIC All header records have this number in c_magic.

CHECKSUM
: Header records checksum to this value.

The fields of the header structure are as follows:

c_type The type of the header.

c_date The date the dump was taken.

c_ddate The date the file system was dumped from.

c_volume The current volume number of the dump.

c_tapea The current number of this (512-byte) record.

c_inumber
: The number of the inode being dumped if this is of type TS_INODE.

c_magic This contains the value MAGIC above, truncated as needed.

c_checksum
: This contains whatever value is needed to make the record sum to CHECKSUM.

c_dinode This is a copy of the inode as it appears on the file system; see filsys(5).

c_count The count of characters in c_addr.

c_addr An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not

sufficient space in this record to describe all of the blocks in a file, TS_ADDR records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure idates describes an entry of the file /etc/ddate where dump history is kept. The fields of the structure are:

id_name The dumped filesystem is `/dev/id_nam'.
id_incno The level number of the dump tape; see dump(1).
id_ddate The date of the incremental dump in system format see types(5).

FILES

/etc/ddate

SEE ALSO

dump(1), dumpdir(1), restor(1), filsys(5), types(5)

## NAME

environ - user environment

## SYNOPSIS

```
extern char **environ;
```

## DESCRIPTION

An array of strings called the `environment' is made available by exec(2) when a process begins. By convention these strings have the form `name=value'. The following names are used by various commands:

PATH The sequence of directory prefixes that sh, time, nice(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by `:'. Login(1) sets PATH=:/bin:/usr/bin.

HOME A user's login directory, set by login(1) from the password file passwd(5).

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as nroff or plot(1), which may exploit special terminal capabilities. See term(7) for a list of terminal types.

Further names may be placed in the environment by the export command and `name=value' arguments in sh(1), or by exec(2). It is unwise to conflict with certain Shell variables that are frequently exported by `.profile' files: MAIL, PS1, PS2, IFS.

## SEE ALSO

exec(2), sh(1), term(7), login(1)

## NAME

filsys, flblk, ino - format of file system volume

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/flbk.h>
#include <sys/filsys.h>
#include <sys/ino.h>
```

## DESCRIPTION

Every file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of blocks. Block Ø is unused and is available to contain a bootstrap program, pack label, or other information.

Block 1 is the super block. The layout of the super block as defined by the include file <sys/filsys.h> is:

```
/*
        (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983.  ALL
        RIGHTS RESERVED.  PROPERTY OF TEXAS INSTRUMENTS INCORPORATED.
        RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE IS SUBJECT
        TO RESTRICTIONS SET FORTH IN TI'S PROGRAM LICENSE AGREEMENT AN
        ASSOCIATED DOCUMENTATION.
*/

#ifdef SCCSID
static char id_filsys[] = "@(#)filsys.hl.2 (Texas Instruments) 83/Ø6/27";
#endif

#ifndef SCCSID

/*      filsys.h 4.3                    81/Ø3/Ø3*/

/*
 * Structure of the super-block
 */
struct filsys
{
        unsigned short s_isize;     /* size in blocks of i-list */
        daddr_t  s_fsize;           /* size in blocks of entire volume */
        short    s_nfree;           /* number of addresses in s_free */
        daddr_t  s_free[NICFREE];   /* free block list */
        short    s_ninode;          /* number of i-nodes in s_inode */
        ino_t    s_inode[NICINOD];  /* free i-node list */
        char     s_flock;           /* lock during free list manipulation */
        char     s_ilock;           /* lock during i-list manipulation */
        char     s_fmod;            /* super block modified flag */
        char     s_ronly;           /* mounted read-only flag */
        time_t   s_time;            /* last super block update */
        daddr_t  s_tfree;           /* total free blocks*/
```

```
        ino_t   s_tinode;           /* total free inodes */
        short   s_dinfo[2];         /* interleave stuff */
#define         s_m                 s_dinfo[0]
#define         s_n                 s_dinfo[1]
        char    s_fsmnt[12];        /* ordinary file mounted on */
        /* end not maintained */
        ino_t   s_lasti;            /* start place for circular search */
        ino_t   s_nbehind;          /* est # free inodes before s_lasti */
};

#ifdef KERNEL
struct filsys *getfs();
#endif

#endif
```

S_isize is the address of the first block after the i-list, which starts just after the super-block, in block 2. Thus is i-list is s_isize-2 blocks long. S_fsize is the address of the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block addresses; if an `impossible' block address is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The s_free array contains, in s_free[1], ... , s_free[s_nfree-1], up to NICFREE free block numbers. NICFREE is a configuration constant. S_free[0] is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in the include file <sys/fblk.h> is:

```
/*
        (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983. ALL
        RIGHTS RESERVED. PROPERTY OF TEXAS INSTRUMENTS INCORPORATED.
        RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE IS SUBJECT
        TO RESTRICTIONS SET FORTH IN TI'S PROGRAM LICENSE AGREEMENT AND
        ASSOCIATED DOCUMENTATION.
*/

#ifdef SCCSID
static char id_fblk[] = "@(#)fblk.h1.2 (Texas Instruments) 83/06/27";
#endif

#ifndef SCCSID

struct fblk
{
        short   df_nfree;
```

```
        daddr_t  df_free[NICFREE];
};

#endif
```

The fields df nfree and df free in a free block are used exactly like s nfree and s free in the super block. To allocate a block: decrement s nfree, and the new block number is s free[s nfree]. If the new block address is Ø, there are no blocks left, so give an error. If s nfree became Ø, read the new block into s nfree and s free. To free a block, check if s nfree is NICFREE; if so, copy s nfree and the s free array into it, write it out, and set s nfree to Ø. In any event set s free[s nfree] to the freed block's address and increment s nfree.

S ninode is the number of free i-numbers in the s inode array. To allocate an i-node: if s ninode is greater than Ø, decrement it and return s inode[s ninode]. If it was Ø, read the i-list and place the numbers of all free inodes (up to NICINOD) into the s inode array, then try again. To free an i-node, provided s ninode is less than NICINODE, place its number into s inode[s ninode] and increment s ninode. If s ninode is already NICINODE, don't bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

S flock and s ilock are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of s fmod on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information. S ronly is a write-protection indicator; its disk value is also immaterial.

S time is the last time the super-block of the file system was changed. During a reboot, s time of the super-block for the root file system is used to set the system's idea of the time.

The fields s tfree, s tinode, s fname and s fpack are not currently maintained.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-nodes are 64 bytes long, so 8 of them fit into a block. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node as given in the include file <sys/ino.h> is:

```
/*
        (C) COPYRIGHT, TEXAS INSTRUMENTS INCORPORATED, 1983.  ALL
        RIGHTS RESERVED.  PROPERTY OF TEXAS INSTRUMENTS INCORPORATED.
        RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE IS SUBJECT
        TO RESTRICTIONS SET FORTH IN TI'S PROGRAM LICENSE AGREEMENT AN
        ASSOCIATED DOCUMENTATION.
*/

#ifdef SCCSID
static char id_ino[] = "@(#)ino.h1.2 (Texas Instruments) 83/06/27";
#endif

#ifndef SCCSID

/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode
{
        unsigned short          di_mode;     /* mode and type of file */
        short   di_nlink;       /* number of links to file */
        short   di_uid;         /* owner's user id */
        short   di_gid;         /* owner's group id */
        off_t   di_size;        /* number of bytes in file */
        char    di_addr[40];    /* disk block addresses */
        time_t  di_atime;       /* time last accessed */
        time_t  di_mtime;       /* time last modified */
        time_t  di_ctime;       /* time created */
};

#endif
```

Di mode tells the kind of file; it is encoded identically to the st mode field of stat(2). Di nlink is the number of directory entries (links) that refer to this i-node. Di uid and di gid are the owner's user and group IDs. Size is the number of bytes in the file. Di atime and di mtime are the times of last access and modification of the file contents (read, write or create) (see times(2)); Di ctime records the time of last modification to the inode or to the file, and is used to determine whether it should be dumped.

Special files are recognized by their modes and not by i-number. A block-type special file is one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the di addr field is occupied by the device code (see types(5)). The device codes of block and character special files overlap.

Disk addresses of plain files and directories are kept in

the array di addr packed into 3 bytes each. The first 10 addresses specify device blocks directly. The last 3 addresses are singly, doubly, and triply indirect and point to blocks of 128 block pointers. Pointers in indirect blocks have the type daddr t (see types(5)).

For block b in a file to exist, it is not necessary that all blocks less than b exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

**SEE ALSO**

icheck(1), dcheck(1), dir(5), mount(1), stat(2), types(5)

**NAME**

group - group file

**DESCRIPTION**

Group contains for each group the following information:

group name
encrypted password
numerical group ID
a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; Each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

**FILES**

/etc/group

**SEE ALSO**

newgrp(1), crypt(3), passwd(1), passwd(5)

## NAME

mtab - mounted file system table

## DESCRIPTION

Mtab resides in directory /etc and contains a table of devices mounted by the mount command. Umount removes entries.

Each entry is 64 bytes long; the first 32 are the null-padded name of the place where the special file is mounted; the second 32 are the null-padded name of the special file. The special file has all its directories stripped away; that is, everything through the last `/' is thrown away.

This table is present only so people can look at it. It does not matter to mount if there are duplicated entries nor to umount if a name cannot be found.

## FILES

/etc/mtab

## SEE ALSO

mount(1)

## NAME

passwd - password file

## DESCRIPTION

Passwd contains for each user the following information:

name (login name, contains no upper case)
encrypted password
numerical user ID
numerical group ID
GCOS job number, box number, optional GCOS user-id
initial working directory
program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The GCOS field is used only when communicating with that system, and in other installations can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

## FILES

/etc/passwd

## SEE ALSO

getpwent(3), login(1), crypt(3), passwd(1), group(5)

NAME

plot - graphics interface

DESCRIPTION

Files of this format are produced by routines described in plot(3), and are interpreted for various devices by commands described in plot(1). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the `current point' for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in plot(3).

**m** move: The next four bytes give a new current point.

**n** cont: Draw a line from the current point to the point given by the next four bytes. See plot(1).

**p** point: Plot the point given by the next four bytes.

**l** line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

**t** label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

**a** arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

**c** circle: The first four bytes give the center of the circle, the next two the radius.

**e** erase: Start another frame of output.

**f** linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are `dotted,' `solid,' `longdashed,' `shortdashed,' and `dotdashed.' Effective only in plot 4014 and plot ver.

**s** space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of plot(1). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face isn't square.

```
4014        space(0, 0, 3120, 3120);
ver         space(0, 0, 2048, 2048);
300, 300s   space(0, 0, 4096, 4096);
450         space(0, 0, 4096, 4096);
```

**SEE ALSO**

plot(1), plot(3), graph(1)

## NAME

termcap - terminal capability data base

## SYNOPSIS

/etc/termcap

## DESCRIPTION

Termcap is a data base describing terminals, used, e.g., by vi(1) and curses(3). Terminals are described in termcap by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in termcap.

Entries in termcap consist of a number of `:' separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by `|' characters. The first name is always 2 characters long and is used by older version 6 systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

## CAPABILITIES

(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

| Name | Type | Pad? | Description |
|---|---|---|---|
| ae | str | (P) | End alternate character set |
| al | str | (P*) | Add new blank line |
| am | bool | | Terminal has automatic margins |
| as | str | (P) | Start alternate character set |
| bc | str | | Backspace if not ^H |
| bs | bool | | Terminal can backspace with ^H |
| bt | str | (P) | Back tab |
| bw | bool | | Backspace wraps from column Ø to last column |
| CC | str | | Command char in prototype if terminal settable |
| cd | str | (P*) | Clear to end of display |
| ce | str | (P) | Clear to end of line |
| ch | str | (P) | Like cm but horiz motion only, line stays same |
| cl | str | (P*) | Clear screen |
| cm | str | (P) | Cursor motion |
| co | num | | Number of columns in a line |
| cr | str | (P*) | Carriage return, (default ^M) |
| cs | str | (P) | Change scrolling region (vtlØØ), like cm |
| cv | str | (P) | Like ch but vertical only. |
| da | bool | | Display may be retained above |
| dB | num | | Number of millisec of bs delay needed |
| db | bool | | Display may be retained below |

| | | | |
|---|---|---|---|
| dC | num | | Number of millisec of cr delay needed |
| dc | str | (P*) | Delete character |
| dF | num | | Number of millisec of ff delay needed |
| dl | str | (P*) | Delete line |
| dm | str | | Delete mode (enter) |
| dN | num | | Number of millisec of nl delay needed |
| do | str | | Down one line |
| dT | num | | Number of millisec of tab delay needed |
| ed | str | | End delete mode |
| ei | str | | End insert mode; give :ei=: if ic |
| eo | str | | Can erase overstrikes with a blank |
| ff | str | (P*) | Hardcopy terminal page eject (default ^L) |
| hc | bool | | Hardcopy terminal |
| hd | str | | Half-line down (forward 1/2 linefeed) |
| ho | str | | Home cursor (if no cm) |
| hu | str | | Half-line up (reverse 1/2 linefeed) |
| hz | str | | Hazeltine; can't print ~'s |
| ic | str | (P) | Insert character |
| if | str | | Name of file containing is |
| im | bool | | Insert mode (enter); give :im=: if ic |
| in | bool | | Insert mode distinguishes nulls on display |
| ip | str | (P*) | Insert pad after character inserted |
| is | str | | Terminal initialization string |
| kØ-k9 | str | | Sent by other function keys Ø-9 |
| kb | str | | Sent by backspace key |
| kd | str | | Sent by terminal down arrow key |
| ke | str | | Out of keypad transmit mode |
| kh | str | | Sent by home key |
| kl | str | | Sent by terminal left arrow key |
| kn | num | | Number of other keys |
| ko | str | | Termcap entries for other non-function keys |
| kr | str | | Sent by terminal right arrow key |
| ks | str | | Put terminal in keypad transmit mode |
| ku | str | | Sent by terminal up arrow key |
| lØ-19 | str | | Labels on other function keys |
| li | num | | Number of lines on screen or page |
| ll | str | | Last line, first column (if no cm) |
| ma | str | | Arrow key map, used by vi version 2 only |
| mi | bool | | Safe to move while in insert mode |
| ml | str | | Memory lock on above cursor. |
| mu | str | | Memory unlock (turn off memory lock). |
| nc | bool | | No working carriage return (DM25ØØ,H2ØØØ) |
| nd | str | | Non-destructive space (cursor right) |
| nl | str | (P*) | Newline character (default \n) |
| ns | bool | | Terminal is a CRT but doesn't scroll. |
| os | bool | | Terminal overstrikes |
| pc | str | | Pad character (rather than null) |
| pt | bool | | Has hardware tabs (may need to be set with is) |
| se | str | | End stand out mode |
| sf | str | (P) | Scroll forwards |
| sg | num | | Number of blank chars left by so or se |
| so | str | | Begin stand out mode |

| | | | |
|---|---|---|---|
| sr | str | (P) | Scroll reverse (backwards) |
| ta | str | (P) | Tab (other than ^I or with padding) |
| tc | str | | Entry of similar terminal - must be last |
| te | str | | String to end programs that use cm |
| ti | str | | String to begin programs that use cm |
| uc | str | | Underscore one char and move past it |
| ue | str | | End underscore mode |
| ug | num | | Number of blank chars left by us or ue |
| ul | bool | | Terminal underlines even though cannot overstrike |
| up | str | | Upline (cursor up) |
| us | str | | Start underscore mode |
| vb | str | | Visible bell (may not move cursor) |
| ve | str | | Sequence to end open/visual mode |
| vs | str | | Sequence to start open/visual mode |
| xb | bool | | Beehive (f1=escape, f2=ctrl C) |
| xn | bool | | A newline is ignored after a wrap (Concept) |
| xr | bool | | Return acts like ce \r \n (Delta Data) |
| xs | bool | | Standout not erased by writing over it (HP 264?) |
| xt | bool | | Tabs are destructive, magic so char (Teleray 1061) |

**A Sample Entry**

The following entry, which describes the Concept-100, is among the more complex entries in the termcap file as of this writing. (This particular concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200:\
   :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :co#80:\
   :dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
   :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and that empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in termcap are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

**Types of Capabilities**

All capabilities have two letter codes. For instance, the fact that the Concept has automatic margins (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character `#' and then the value. Thus **co** which indicates the number of columns the

terminal has gives the value `80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an `=', and then a string ending at the next following `:'. A delay in milliseconds may appear after the `=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. `20', or an integer followed by an `*', i.e. `3*'. A `*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a `*' is specified, it is sometimes useful to give a delay of the form `3.5' specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a ****, and the characters **^** and **** may be given as **\^** and ****. If it is necessary to place a **:** in a capability it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability it must be encoded as **\200**. The routines which deal with termcap use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

**Preparing Descriptions**

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in termcap and to build up a description gradually, using partial descriptions with ex to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the termcap file to describe it or bugs in ex. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in /etc/termcap. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

**Basic capabilities**

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT,

then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than ^H (ugh) in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in termcap are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the termcap file usually assumes that this is on, i.e. **am**.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

**Cursor addressing**

Cursor addressing in the terminal is described by a **cm** string capability, with printf(3s) like escapes **%x** in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the **%** encodings have the following meanings:

```
%d    as in printf, 0 origin
%2    like %2d
%3    like %3d
%.    like %c
%+x   adds x to value, then %.
%>xy  if value > x adds y, no output.
%r    reverses order of line and column, no output
%i    increments line/column (for 1 origin)
```

%% gives a single %
%n exclusive or row and column with Ø14Ø (DM25ØØ)
%B BCD (16*(x/1Ø)) + (x%1Ø), no output.
%D Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12cØ3Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is cm=6\E&%r%2c%2Y. The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, cm=^T%.%.. Terminals which use %. need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** ^D and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cm=\E=%+ %+ .

**Cursor motions**

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

**Area clears**

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

**Insert/delete line**

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal

can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

**Insert/delete character**

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using termcap. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not not falling into one of these two classes.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as **ei** the sequence to leave insert mode (give this, with an empty value also if you gave **im** so). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen

if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

**Highlighting, underlining, and visible bells**

If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining - half bright is not usually an acceptable standout mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1Ø61 do, this is acceptable, and although it may confuse some programs slightly, it can't be helped.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of ex, this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for

example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

**Keypad**

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the termcap 2 letter codes can be given in the **ko** capability, for example, :ko=cl,ll,sf,sb:, which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

**Miscellaneous**

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, then this can be given as **ta**.

Hazeltine terminals, which don't allow `~' characters to be printed should indicate **hz**. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc**. Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is /usr/lib/tabset/std but **is** clears the tabs first.

**Similar Terminals**

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be last and the combined length of the two entries must not exceed 1024. Since termlib routines search the entry from left to right, and since the **tc** capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with **xx@** where xx is the capability. For example, the entry

```
hn|2621nl:ks@:ke@:tc=2621:
```

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

**FILES**

/etc/termcap file containing terminal descriptions

**SEE ALSO**

ex(1), curses(3), termcap(3), tset(1), vi(1), ul(1), more(1)

**AUTHOR**

William Joy

Mark Horton added underlining and keypad support

## BUGS

Ex allows only 256 characters for string capabilities, and the routines in termcap(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the vi program.

Not all programs support all entries. There are entries that are not supported by any program.

## NAME

ttys - terminal initialization data

## DESCRIPTION

The ttys file is read by the init program and specifies which terminal special files are to have a process created for them which will allow people to log in. It contains one line per special file.

The first character of a line is either `Ø' or `1'; the former causes the line to be ignored, the latter causes it to be effective. The second character is used as an argument to getty(8), which performs such tasks as baud-rate recognition, reading the login name, and calling login. For normal lines, the character is `Ø'; other characters can be used, for example, with hard-wired terminals where speed recognition is unnecessary or which have special characteristics. (Getty will have to be fixed in such cases.) The remainder of the line is the terminal's entry in the device directory, /dev.

## FILES

/etc/ttys

## SEE ALSO

init(8), getty(8), login(1)

**NAME**

types - primitive system data types

**SYNOPSIS**

```
#include <sys/types.h>
```

**DESCRIPTION**

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
/*
```



```
*/

#ifdef SCCSID
static char id_types[] = "@(#)types.h   1.2 (Texas Instruments) 83/06/27";
#endif

#ifndef SCCSID

typedef   long       daddr_t;
typedef char *       caddr_t;
typedef   long       mem_t;
typedef   unsigned short ino_t;
typedef   long       time_t;
typedef   long       label_t[13];    /* regs d2-d7, a2-a7, pc */
typedef   short           dev_t;
typedef   long       off_t;

     /* selectors and constructor for device code */
#define   major(x)        (int)(((unsigned)x>>8))
#define   minor(x)        (int)(x&0377)
#define   makedev(x,y)    (dev_t)((x)<<8|(y))

#endif
```

The form daddr_t is used for disk addresses except in an i-node on disk, see filsys(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The label_t variables are used to save the processor state while another process is running.

**SEE ALSO**

filsys(5), time(2), lseek(2), adb(1)

## NAME

utmp, wtmp - login records

## SYNOPSIS

**#include <utmp.h>**

## DESCRIPTION

The utmp file allows one to discover information about who is currently using UNIX. The file is a sequence of entries with the following structure declared in the include file:

```
struct utmp {
     char ut_line[8];          /* tty name */
     char ut_name[8];          /* user id */
     long ut_time;        /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of time(2).

The wtmp file records all logins and logouts. Its format is exactly like utmp except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name `~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names `|' and `}' indicate the system-maintained time just before and just after a date command has changed the system's idea of the time.

Wtmp is maintained by login(1) and init(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by ac(1).

## FILES

/etc/utmp
/usr/adm/wtmp

## SEE ALSO

login(1), init(8), who(1), ac(1)