Command Reference

IBM Personal Computer XENIX™ Operating System

Programming Family

Personal
Computer
Software

6138656

Section 1. XENIX Operating System Commands

Introduction to (C) Commands

intro - Introduces XENIX commands.

Description

This section describes use of the individual commands available in the XENIX Operating System.

Syntax

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

name [*options*] [*cmdargs*]

where:

name	Is the name of an executable file and must be entered exactly as shown.
option	*-noargletters* or, *-argletter<>optarg* where <> is optional spaces.
noargletter	Is a single letter representing an option without an argument.
argletter	Is a single letter representing an option requiring an argument.

optarg	Is an argument (character string) satisfying preceding *argletter*
cmdarg	Is a path name (or other command argument) *not* beginning with a hyphen. A hyphen by itself indicates the standard input.

See Also

getopt(C), getopt(S)

Diagnostics

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

Comment

Not all commands require options and arguments.

ACCTCOM(C)

Name

acctcom - Searches for and prints process accounting files.

Syntax

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

Description

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

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

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

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

The **options** are:

-b	Reads backward, showing latest commands first.
-f	Prints the *fork / exec* flag and system exit status columns in the output.
-h	Instead of mean memory size, shows the fraction of total available CPU time consumed by the process during its execution. This "hogging factor" is computed as: (total CPU time)/(elapsed time).
-i	Prints columns containing the I/O counts in the output.
-k	Instead of memory size, shows total kcore-minutes.
-m	Shows mean core size (the default).
-r	Shows CPU factor (user time/(system-time + user-time))
-t	Shows separate system and user CPU times.
-v	Excludes column headings from the output.
-l *line*	Shows only processes belonging to terminal /**dev**/ **line**
-u *user*	Shows only processes belonging to *user* that may be specified by a user ID, a login name that is then converted to a user ID, a #, which designates only the processes executed with super-user privileges, or ?, which designates only those processes associated with unknown user IDs.
-g *group*	Shows only processes belonging to *group.* The **group** may be designated by either the group ID or group name.
-d *mm/dd*	Any *time* arguments following this flag are assumed to occur on the given month and day, rather than during the last 24 hours. This is needed for looking at old files.

-s *time*	Shows only the processes that existed on or after *time,* given in the form **hr:min:sec**. The **:sec** or **:min:sec** may be omitted.
-e *time*	Shows only the processes that existed on or before *time* . Using the same *time* for both **-s** and **-e** shows the processes that existed at *time.*
-n *pattern*	Shows only commands matching *pattern* that may be a regular expression as in **ed**(C) except that + means one or more occurrences.
-H *factor*	Shows only processes that exceed *factor* , where factor is the "hogging factor" as explained in option **-h** above.
-O *time*	Shows only the processes with operating system CPU time that exceeds *time.*
-C *time*	Shows only the processes that exceed *time* (the total CPU time).

Multiple options have the effect of a logical AND.

Files

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

See Also

accton(C), ps(C), su(C), acct(F), utmp(M)

Comment

The **acctcom** command only reports on processes that have terminated; use **ps**(C) for active processes.

ACCTON(C)

Name

accton - Turns on accounting.

Syntax

```
accton [file]
```

Description

The **accton** command turns on and off process accounting. If no *file* is given, accounting is turned off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records. (see **acct(F)**).

Files

/etc/passwd	Used for login name to user ID conversions
/usr/lib/acct	Holds many accounting commands
/usr/adm/pacct	Current process accounting file
/usr/adm/wtmp	Login/logout history file

See Also

acctcom(C), acct(F), utmp(M)

ASKTIME(C)

Name

asktime - Prompts for the correct time of day.

Syntax

```
/etc/asktime
```

Description

This command prompts for the time of day. You must enter a legal time according to the proper format as defined below:

[*yymmdd*] *hhmm*

Here *yy* is the last two digits of the year number; the first *mm* is the month number; *dd* is the day number in the month. The date is optional. The current year, month, and day is the default if you do not enter any date. The *hh* is the hour number (24-hour system); the second *mm* is the minute number.

Examples

This example sets the new time, date, and year to "11:29 April 20, 1984":

```
current System Time is Wed Nov  3 14:36:23 PST 1982
Enter time ([yymmdd] hhmm): 8404201129
```

Diagnostics

If you enter an illegal time, **asktime** displays:

```
cvtdate: bad conversion
```

and exits.

Comment

The **asktime** command is normally performed automatically by the system startup file **/etc/rc** immediately after the system is booted; however, it may be executed at any time. The command is privileged, and can only be executed by the super-user.

ASSIGN(C)

Name

assign, deassign - Assigns and deassigns devices.

Syntax

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

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

Description

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

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

Available options:

-d Performs the action of **deassign**.

-v Gives verbose output.

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

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

```
assign fd
```

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

Files

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

Diagnostics

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

Comments

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

The first time **assign** is invoked it builds the assignable devices table **/etc/atab**. This table is used in subsequent invocations to save repeated searches of the **/dev** directory. If one of the devices in **/dev**, is changed to assignable (that is, *asg* owns the device), the super-user must remove **/etc/atab** so that a correct list will be built the next time the command is invoked.

AT(C)

Name

at, atq, atrm - Executes commands at a later time.

Syntax

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

atq [**-l**]

atrm *idnumber* . . .

Description

The **at** command causes the contents of a file to be executed as a shell script at a specified time. This command is useful for running processes at regular intervals or when the system is not busy.

The arguments are:

time	One to four digits, followed by an optional "a" for am, "p" for pm, "n" for noon, or "m" for midnight. One- and two-digit numbers are interpreted as hours, three- and four-digit numbers as hours and minutes. If no letters follow the digits, 24-hour time is assumed.
day	Either a month name followed by a day number, or the name of a day of the week. If the word "week" follows the name of the day, the file is invoked seven days after the day named. Names of months and days may be recognizably truncated. (See the Examples later in this section.)
file	The name of the file containing the commands to be executed. If no *file* is specified, the standard input is assumed.

The **at** command creates a file that is executed by the shell at the specified time. This file contains a comment line that lists the user's user ID and group ID, a **cd** command that changes the working directory of the process to the one you were using when you executed **at**, assignments to the appropriate environment variables, and the *file* specified in the **at** command line. Output from processes in **file** must be redirected or (on most systems) it is lost. The **at** command's shell scripts are run by periodic execution of the command **/usr/lib/atrun** from **cron**(C)

The **atq** command gives the following information about files waiting to be processed:

- The user ID under which the file will run
- A unique ID number used to reference the file
- The date and time the file will be processed

The **-l** option displays the commands in each file in the queue.

The **atrm** command removes files from the "at" queue. The **atrm** command uses the ID numbers from the **atq** command to remove the specific files. A user can only remove his own files.

Examples

Use the following line to place a file in the queue:

at 8a jan 24 *file*

In the following command line, *file* will be executed a week from this Friday at 3:30p.m.

at 1530 fr week *file*

To remove a file from the queue, find out the ID numbers with:

atq

Then remove the file with **atrm**:

atrm *idnumber*

Files

/usr/spool/at/yy.ddd.hhhh.uu
: Activity to be performed at hour **hhhh** of 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
: Contains old **at** files.

/usr/lib/atrun
: Program that executes activities at the specified time.

See Also

calendar(C), cron(C), pwd(C)

Diagnostics

Points out about various syntax errors and times out of range.

Comments

The directory **/usr/spool/at/past** should be periodically emptied by the super-user.

Because of the granularity of the execution of **/usr/lib/atrun** , there may be problems in scheduling things exactly 24 hours into the future.

AWK(C)

Name

awk - Searches for and processes a pattern in a file.

Syntax

---

awk [-F*c*][-**f** *programfile*+ | '*program*'][*file* . . .]

Description

The **awk** command scans each input *file* for lines that match patterns specified in *program* or in *programfile*. When a line of *file* matches a pattern, an associated action may be performed. This command is useful for compiling information, performing arithmetic on input data, and for doing iterative or conditional processing.

The options are:

-F*c* Sets the field separator variable (FS) to the letter "c". The default field separators are tab and space.

-f Causes **awk** to take its program from *programfile* .

The arguments are:

programfile
A file containing an **awk** program.

program An **awk** program. Programs given on the command line must be enclosed in single quotation marks to prevent interpretation by the shell.

file . . .
The names of the files to be processed. If no filename is given, the standard output is used.

An **awk** program consists of statements in the form:

pattern {*action*}

Pattern-action statements may appear on the **awk** command line, or in an **awk** program file.

If no *pattern* is given, all lines in the input file are matched. If no *action* is given, each matched line is displayed on the standard output.

A pattern may be a literal string or a regular expression, or a combination of a regular expression and a field or variable separated by operators.

The **awk** command also provides two patterns, BEGIN and END, that can be used to perform actions before the first line is read, and after the last line is read, respectively.

To select a range of lines, use two patterns on a single program line, separated by a comma.

An action is a sequence of statements separated by a semicolon, newline, or right brace. See "Statements" later in this section.

Variables

In addition to variables declared and initialized by the user, **awk** has the following program variables:

NR	Number of records
NF	Number of fields in a record
FS	Input field separator
OFS	Output field separator
RS	Input record separator

ORS	Output record separator
$Ø	The current record
$1, $n	Fields in the current record
OFMT	The output format for numbers. The default is %.6g
FILENAME	The name of the input file currently being read.

Arrays may be used to store data. Arrays do not need to be dimensioned before use. For example, w[i] denotes the ith item of array w.

Expressions

A pattern match with a field or variable may be tested with the following operators:

~	Matches the regular expression.
!~	Does not match the regular expression.

The **awk** command processes relational expressions using the following operators:

<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
>=	Greater than or equal to
>	Greater than

Patterns can be combined using the operators:

&&	And
||	Or
!	Not

An empty expression list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the following operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

Concatenation is indicated by a blank. The following C operators are also available in expressions:

++	Increment
--	Decrement
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign

Statements

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

if	Used the same as in the C language.
while	Used the same as in the C language.
for	The iterative construction. It can be used the same as in the C language, or as an array iterator.
break	Similar to its C counterpart.
continue	Similar to its C counterpart.
print	Prints its arguments on the standard output or in a file if redirected.
printf	Prints *expression-list* in the format specified in *format.*
next	Stops processing the current record and moves to the next record, if any.

Comments are preceded by a number sign (#).

Functions

The **awk** command has the following built-in functions:

exit(*x*)	Terminates the **awk** program. If *x* is given, this value is **awk's** return value. If *x* is not given, 0 is returned. If the program has an END section, it is invoked before termination.

exp(*x***)**	Exponentiation of the value of *x*.
index(*s, t***)**	Returns the starting position of the leftmost occurrence of *t* in *s*. If *t* is not a substring of *s*, then index(*s*,*t*) is∅.
int(*x***)**	Returns the largest integer less than or equal to *x*. If *x* is negative, its value is the smallest integer greater than or equal to *x*.
length(*x***)**	A function whose value is the number of characters in the string (*x*). With no arguments **length** is equivalent to $0.
log(*x***)**	Natural logarithm of *x*.
split(*x, y***)**	Assigns the fields of string *x* to successive elements of array *y*.
sqrt(*x***)**	Square root of *x*.
substr(*string, index, length***)**	Returns the substring of *string* that begins at *index* and is *length* characters long.

Examples

The following displays lines in *file* longer than 72 characters:

awk 'length > 72' *file*

The following prints the first two fields in opposite order:

awk '{ print $2, $1 }' *file*

BACKUP(C)

Name

backup - Performs incremental file system backup.

Syntax

```
backup [ key [ arguments ] filesystem ]
```

Description

The **backup** command copies to the specified device all files changed after a certain date in the *filesystem*. The *key* specifies the date and other options about the backup, where a *key* consists of characters from the set **0123456789kfusd**. The meanings of these characters are described below:

0-9 This number is the "backup level". Backs up all files modified since the last date stored in the file **/etc/ddate** for the same file system at lesser levels. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire file system to be backed up.

k This option is used when backing up to a block-structured device, such as a diskette. The size (in K-bytes) of the volume being written is taken from the next *argument*. If the **k** argument is specified, any **s** and **d** arguments are ignored. The default is to use **s** and **d**.

f Places the backup on the next *argument* file instead of the default device.

u If the backup completes successfully, writes the date of the beginning of the backup to the file **/etc/ddate**. This file records a separate date for each file system and each backup level.

s For backups to magnetic tape, the size of the tape specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, **backup** waits for reels to be changed. The default size is 2,300 feet.

d For backups to magnetic tape, 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 backed up to the default device.

The first backup should be a full level-0 backup:

```
backup 0u
```

Next, periodic level-9 backups should be made on an exponential progression of tapes or diskettes:

```
backup 9u
```

(This is sometimes called the Tower of Hanoi progression after the name of the game where a similar progression occurs, for example, 1 2 1 3 1 2 1 4 . . . where backup 1 is used every other time, backup 2 every fourth, backup 3 every eighth, etc.) When the level-9 incremental backup becomes unmanageable because a tape is full or too many diskettes are required, a level-1 backup should be made:

```
backup 1u
```

After this, the exponential series should progress as if uninterrupted. These level-9 backups are based on the level-1 backup, which is based on the level-0 full backup. This progression of levels of backups can be carried as far as desired.

The default file system and the backup device depend on the settings of the variables DISK and TAPE, respectively, in the file **/etc/default/backup**.

Files

/etc/ddate	Records backup dates of file system/level
/etc/default/backup	Default backup information

See Also

cpio(C), default(M), dumpdir(C), restore(C), tar(C), backup(F)

Diagnostics

If the backup requires more than one volume (where a volume is likely to be a diskette or tape), you will be asked to change volumes. Press Enter after changing volumes.

Comments

Sizes are based on 1600 BPI for blocked tape; the raw magnetic tape device has to be used to approach these densities. Write errors to the backup device are usually irrecoverable. Read errors on the file system are ignored.

> **Warning:** When backing up to diskettes, be sure to have enough formatted diskettes ready before starting a backup.

BANNER(C)

Name

banner - Prints large letters.

Syntax

```
banner strings
```

Description

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

BASENAME(C)

Name

basename - Removes directory names from path names.

Syntax

basename *string* [*suffix*]

Description

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

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

Examples

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

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

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

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

See Also

dirname(C), sh(C)

BC(C)

Name

bc - Invokes a calculator.

Syntax

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

Description

The **bc** program is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input.

The **bc** program is actually a preprocessor for **dc**(C), which it invokes automatically, unless the **-c** (compile only) option is present. If the **-c** option is present, the **dc** input is sent to the standard output instead. The **-l** argument stands for the name of an arbitrary precision math library. The syntax for **bc** programs is: L means the letters a-z, E means expression, S means statement.

Comments

Enclosed in /* and */

Names

Simple variables: L

Array elements: L [E]

The words "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)

Additive operators:

+	Addition
-	Subtraction

Multiplicative operators:

*	Multiplication
/	Division
%	Modulo (remainder)
^	Exponentiation

Unary operators:

++	Increment
--	Decrement (prefix and postfix; apply to names)

Relational operators:

==	Equal to
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to
<	Less than
>	Greater than

Assignment operators:

=	Assign
=+	Add and assign
=-	Subtract and assign
=*	Multiply and assign
=/	Divide and assign
=%	Modulo and assign
=^	Exponentiate and assign

Statements:

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

Function definitions:

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

Functions in -l math library:

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

All function arguments are passed by value.

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

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

Example

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

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

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

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

Files

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

See Also

dc(C)
bc in the IBM Personal Computer *XENIX Basic Operations Guide.*

Comments

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

A *quit* is interpreted when read, not when executed.

BDIFF(C)

Name

bdiff - Compares files too large for diff.

Syntax

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

Description

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

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

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

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

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

Files

/tmp/bd?????

See Also

diff(C)

Diagnostics

Use help(CP)

Comment

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

BFS(C)

Name

bfs - Scans big files.

Syntax

```
bfs [-]name
```

Description

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

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

All address expressions described under **ed** are supported. In addition, regular expressions may be surrounded with two symbols other than the standard slash (**/**) and question mark (**?**). A greater-than sign (**>**) indicates downward search without wraparound, and a less-than sign (**<**) indicates upward search without wraparound. Because **bfs** uses a different regular expression-matching routine than **ed**, the regular expressions accepted are slightly wider in scope. The differences from **ed** syntax include the fact that parentheses and braces are special and need not be escaped.

Differences are listed below:

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

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

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

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

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

The **e** , **g** , **v** , **k** , **n** , **p** , **q** , **w** , **=** , **!** and null commands operate as described under **ed** . Commands such as **---**, +++-, +++=, **-12**, and **+4p** are accepted. Note that **1,10p** and **1,10** both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no remembered filename. The **w** command is independent of output diversion, truncation, or crunching (see the **xo, xt and xc** commands, below).

The following additional commands are available:

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

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

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

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

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

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

2. The second address is less than the first.

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

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

```
xb/^/ label
```

is an unconditional jump.

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

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

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

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

prints the first 100 lines.

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

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

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

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

Another feature of the **xv** command is that the first line of output from a XENIX command can be stored into a variable. The only requirement is that the first character of *value* be a **!** For example:

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

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

```
xv7\!date
```

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

xbz *label*

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

```
xv55
: 1
/size/
xv5!expr %5 - 1
!if Ø%5 != Ø exit 2
xbn 1

xv45
: 1
/size/
xv4!expr %4 - 1
!if Ø%4 = Ø exit 2
xbz 1
```

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

See Also

csplit(C), ed(C)

Diagnostics

If prompting is turned off, a question mark (**?**) is printed for errors in commands. When prompting is on, self-explanatory error messages appear.

CAL(C)

Name

cal - Prints a calendar.

Syntax

```
cal [ [month] year]
```

Description

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

Comments

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

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

CALENDAR(C)

Name

calendar - Invokes a reminder service.

Syntax

```
calendar [ - ]
```

Description

The **calendar** command consults the file *calendar* in the user's current directory and prints out lines that contain today's or tomorrow's date. Most reasonable month-day dates, such as "Sep. 7", "September 7", and "9/7", are recognized, but not "7 September", "7/12" or "07/12".

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

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

Files

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

See Also

cron(C), mail(C)

Comment

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

CAT(C)

Name

cat - Concatenates and displays files.

Syntax

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

Description

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

-s	Suppresses warnings about nonexistent files.
-u	Causes the output to be unbuffered.

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

Examples

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

```
cat file
```

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

```
cat file1 file2 > file3
```

The following example appends a copy of *file1* to *file2*:

```
cat file1 >> file2
```

See Also

cp(C), pr(C)

CD(C)

Name

cd - Changes working directory.

Syntax

```
cd [directory]
```

Description

The **cd** command changes the current directory to the directory specified by *directory*. If no *directory* is specified, the value of the shell parameter $HOME is used, and the user is placed in his home directory. The argument ".." moves up from a directory to the parent directory.

The user must have search (execute) permission in all directories specified in *directory*.

Examples

The following example changes the current directory to the user's home directory:

```
cd
```

The following command changes the current directory to **/usr/joe/memos/meetings**:

```
cd /usr/joe/memos/meetings
```

The following command changes the current directory from **/usr/joe/memos/meetings** to **/usr/joe/accounts/overdue**:

```
cd ../../accounts/overdue
```

See Also

chroot(C), pwd(C), sh(C)

CHGRP(C)

Name

chgrp - Changes group ID.

Syntax

chgrp *group file* . . .

Description

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

Files

/etc/passwd
/etc/group

See Also

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

Comment

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

CHMOD(C)

Name

chmod - Changes the access permissions of a file or directory.

Syntax

chmod *mode file* . . .

Description

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

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

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

Who is one or any combination of the following letters:

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

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

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

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

The operators are:

+ Adds permission

- Removes permission

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

Permissions can be any combination of the following letters:

x	Execute (search permission for directories)
r	Read
w	Write
s	Sets owner or group ID on execution of the file to that of the owner of the file. The mode "u+s" sets the user ID bit for the file. The mode "g+s" sets the group ID bit. Other combinations have no effect.
t	Saves text in memory upon execution. Only the mode "u+t" sets the sticky bit. All other combinations have no effect. This mode can only be set by the super-user.

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

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

chmod *mode filename*

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

4000	Set user ID on execution
2000	Set group ID on execution
1000	Sets the sticky bit
0400	Read by owner
0200	Write by owner
0100	Execute (search in directory) by owner
0040	Read by group

0020	Write by group
0010	Execute (search in directory) by group
0004	Read by others
0002	Write by others
0001	Execute (search in directory) by others
0000	No permissions

Examples

Symbolic Mode

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

```
chmod +x file
```

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

```
chmod go-rw file
```

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

```
chmod o+rw file
```

The following command gives read permission to group and other:

```
chmod g+r,o+r file
```

Absolute Mode

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

```
chmod 0777 file
```

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

```
chmod 0666 file
```

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

```
chmod 0600 file
```

See Also

ls(C)

Comments

The user's *umask* may affect the default settings.

The user ID, group ID, and sticky bit settings, are only useful for binary executable files. They have no effect on shell scripts.

CHOWN(C)

Name

chown - Changes owner ID.

Syntax

```
chown owner file . . .
```

Description

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

Files

/etc/passwd
/etc/group

See Also

chgrp(C), group(M), passwd(M)

Comment

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

CHROOT(C)

Name

chroot - Changes root directory for command.

Syntax

chroot *newroot command*

Description

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

Notice that:

```
chroot newroot command >x
```

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

This command is restricted to the super-user.

The new root path name is always relative to the current root even if a **chroot** is currently in effect. The *newroot* argument is relative to the current root of the running process. Note that it is not possible to change directories to what was formerly the parent of the new root directory; that is, the **chroot** command supports the new root as an absolute root for the duration of the *command*. This means that /.. is equivalent to / everytime.

Comment

Exercise extreme caution when referencing special files in the new root file system.

CHSH(C)

Name

chsh - change user default shell

Syntax

chsh

Description

Use **chsh** to change the default shell assigned to a user. It is the preferred method for updating the default user shell because it handles all the necessary file updates. and validates whether the desired shell is available.

The program prompts for the user login ID and validates that the ID exists in the password file. If so, the requestor is asked to choose a default shell. The choices are **sh** (for the Bourne shell), **csh** (for the C-shell), and **vsh** (for the visual shell). The C-shell option will only be permitted if the Software Development System is installed. When a valid choice is made, **chsh** will insure that appropriate shell files are placed in the user home directory (if they were not already present). For the Borne shell, a *.profile* file is created. For the C-shell, *.login* and *.cshrc* files are created. The **chsh** command can only be executed by the super-user.

Files

/usr/lib/mkuser/mkuser.prof /usr/lib/mkuser/mkuser.login /usr/lib/mkuser/mkuser.cshrc /etc/passwd

See Also

passwd(C), pwadmin(C)

CMP(C)

Name

cmp - Compares two files.

Syntax

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

Description

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

The options are:

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

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

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

See Also

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

Diagnostics

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

COMM(C)

Name

comm - Selects or rejects lines common to two sorted files.

Syntax

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

Description

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

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

See Also

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

COPY(C)

Name

copy - Copies groups of files.

Syntax

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

Description

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

If files, directories, or special files do not exist at the destination, they are created with the same modes and flags as the source. In addition, the super-user may set the user and group ID. The owner and mode are not changed if the destination file exists. Note that there may be more than one source directory. If so, the effect is the same as if the **copy** command had been issued for each source directory with the same destination directory for each copy.

All of the options must be given as separate arguments, and they may appear in any order even after the other arguments. The arguments are:

-a	Asks the user before attempting a copy. If the response does not begin with a y, a copy is not done. This option also sets the **-ad** option.
-l	Uses links instead whenever they can be used. Otherwise a copy is done. Note that links are never done for special files or directories.

-n	Requires the destination file to be new. If not, the **copy** command does not change the destination file. The **-n** flag is meaningless for directories. For special files an **-n** flag is assumed (that is, the destination of a special file must not exist).
-o	If set, every file copied has its owner and group set to those of the source. If not set, the file's owner is the user who invoked the program.
-m	If set, every file copied has its modification time and access time set to that of the source. If not set, the modification time is set to the time of the copy.
-r	If set, every directory is recursively examined as it is encountered. If not set, any directories found are ignored.
-ad	Asks the user whether an **-r** flag applies when a directory is discovered. If the answer does not begin with a y, the directory is ignored.
-v	If the verbose option is set, messages are printed that reveal what the program is doing.
source	This may be a file, directory or special file. It must exist. If it is not a directory, the results of the command are the same as for the **cp** command.
dest	The destination must be either a file or directory that is different than the source.

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

Comment

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

CP(C)

Name

cp - Copies files.

Syntax

cp *file1 file2*

cp *files directory*

Description

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

See Also

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

Comment

Special device files can be copied. If the file is a named pipe, the data in the pipe is copied to a regular file. Similarly, if the file is a device, the file is read until the end-of-file is reached and that data is copied to a regular file. It is illegal to copy a directory to a file.

CPIO(C)

Name

cpio - Copies file archives in and out.

Syntax

cpio -o[**acBv**]

cpio -i[**Bcdmrtuv**] [*patterns*]

cpio -p[**adlmuv**] *directory*

Description

The **cpio -o** (copy out) command 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.

The **cpio -i** (copy in) command 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**(C). In *patterns*, the special characters **?** ***** and [**. . .**] match the slash (/) character. The default for *patterns* is ***** (that is, select all files).

Remember to escape special characters to prevent expansion by the shell.

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

The meanings of the available options are:

-a Resets access times of input files after they have been copied.

-B Blocks input/output 5120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from raw devices).

-d Directories are created as needed.

-c Writes header information in ASCII character form for portability.

-r Interactively renames files. If the user types a null line, the file is skipped.

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

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

-v Verbose: causes a list of filenames to be printed. When used with the **-t** option, the table of contents looks like the output of an **ls -l** command (seels(C)).

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

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

Examples

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

```
ls | cpio -o >/dev/fdØ
```

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

Or:

```
find . -print | cpio -oB >/dev/rfdØ"
```

See Also

backup(C), find(C), tar(C), backup(F), cpio(F) p.

Comment

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.

CRON(C)

Name

cron - Executes commands at specified times.

Syntax

```
/etc/cron
```

Description

The **cron** clock daemon 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**.

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

- A number in the (respective) range indicated above
- Two numbers separated by a hyphen (indicating an inclusive range)
- A list of numbers separated by commas (meaning all of these numbers)
- An asterisk (meaning all legal values)

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

The **cron** command examines **crontab** periodically to see if it has changed; if it has, **cron** reads it. Thus it takes only a short while for entries to become effective.

Example

An example **crontab** file follows:

```
30 4 * * *      /etc/sa -s > /dev/null
0  4 * * *      calendar -
15 4 * * *      find /usr/preserve -mtime +7 -a -exec rm -f {} \;
30 4 1 1 1      /usr/lib/uucp/cleanlog
40 4 * * *      find / -name '#*' -atime +3 -exec rm -f {} \;
0,5,10,15,20,25,30,35,40,45,50,55 * * * */usr/lib/atrun
0,10,20,30,40,50 * * * * /etc/dmesg - >>/usr/adm/messages
1,21,41 * * * * (echo -n ' '; date; echo) >/dev/console
```

A history of all actions by **cron** can be recorded in **/usr/lib/cronlog.** This logging occurs only if the variable CRONLOG in **/etc/default/cron** is set to YES. By default, this value is set to NO and no logging occurs. If logging is turned on, be sure to monitor the size of **/usr/lib/cronlog** so that it doesn't unreasonably consume disk space.

Files

/usr/lib/crontab
/usr/lib/cronlog
/etc/default/cron

See Also

sh(C)

Comment

The **cron** command reads **crontab** only when it has changed, but it reads the in-core version of that table periodically.

CSPLIT(C)

Name

csplit - Splits files according to context.

Syntax

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

Description

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

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

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

.

.

.

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

The options to *csplit* are:

-s	Normally, **csplit** prints the character counts for each file created. If the **-s** option is present, **csplit** suppresses the printing of all character counts.
-k	Normally, **csplit** removes created files if an error occurs. If the **-k** option is present, **csplit** leaves previously created files intact.

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

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

/rexp/	A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp.* The current line becomes the line containing *rexp.* This argument may be followed by an optional + or - some number of lines (for example, /Page/-5).
%rexp%	This argument is the same as /rexp/, except that no file is created for the section.
lnno	A file is to be created from the current line up to (but not including) *lnno.* The current line becomes *lnno.*
{*num*}	Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno,* the file will be split every *lnno* lines (num times) from that point.

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

Examples

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

This example creates four files, **cobolØØ** . . . **cobolØ3**. After editing the "split" files, they can be recombined as follows:

```
cat cobolØ[Ø-3] > file
```

Note that this example overwrites the original file.

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

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

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

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

See Also

ed(C), sh(C)

Diagnostics

Self-explanatory except for:

```
arg - out of range
```

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

Name

cu - Calls another XENIX system.

Syntax

cu [**-s***speed*] [**-a***acu*] [**-l***line*] [**-h**] [**-o** | **-e**] *telno*

cu [**-s***speed*] **-l***line* [**-h**] [**-o** | **-e**] **dir**

Description

The **cu** command "calls up" another XENIX system through a modem or a direct serial connection. It also controls the transmission and reception of data and programs during the call. The **cu** command looks at each line in the file **/usr/lib/uucp/L-devices** until it finds a line that matches the options given in the command line. If it finds an appropriate line, it will attempt to make a connection. If it cannot find the proper line, **cu** quits.

The options are:

-s*speed* Specifies the transmission speed. 1200 baud is the default value. Other speeds available are 110, 150, 300, 1200, 2400, 4800 and 9600 baud. Directly connected machines may by set to other speeds. Most modems are restricted to 300 and 1200 baud. Note, speeds higher than 2400 baud may cause transmission errors.

-a*acu* Specifies the device name of the ACU (automatic calling unit) device. When used with the **-l** option, overrides the search for the first available ACU with the right speed.

-l*line* Specifies the device name of the communications line. When used with the **-a** option, overrides the search for the first available ACU with the right speed.

-h Emulates local echo. This feature supports calls to systems that expect half-duplex mode terminals.

-e Specifies that even-parity data is to be generated for data sent to the remote system.

-o Specifies that odd-parity data is to be generated for data sent to the remote system.

The telephone number of the remote system is *telno*. A comma indicates a delay at appropriate places, for example, to wait for a secondary dial tone when calling from an internal phone system. For directly connected lines, the string **dir** is used instead of *telno*. Direct lines require a line to be specified, but no ACU. See the "Examples" in this section for sample command lines.

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

The *transmit* process interprets lines beginning with a tilde as follows:

~.

Terminates the conversation.

~!

Escapes to an interactive shell on the local system.

~!cmd . . .

Runs **cmd** on the local system (via **sh** -c).

~$cmd . . .

Runs **cmd** locally and sends its output to the remote system.

~%take *remote* [*local*]

Copies file *remote* (on the remote system) to file *local* on the local system. If *local* is omitted, the *remote* filename is used in both places. Use of this line requires the existence of **echo**(C) and **cat**(C) on the remote system. If tabs are to be copied without expansion, **stty tabs** mode should be set on the remote system.

~%put *local* [*remote*]

Copies file *local* (on the local system) to file *remote* on the remote system. If *remote* is omitted, the *local* filename is used in both places. Use of this line requires the existence of **stty**(C) and **cat**(C) on the remote system. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

~~ . . .

Sends the line ~ . . . to the remote system.

~%nostop

Turns off the XON/XOFF input control protocol for the remainder of the session. This is useful if the remote system is one that does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. A line from the remote system that begins with ~> diverts the output to a file. Data is appended to a file if ~>> is used. The diversion is terminated by a trailing ~> . The complete sequence is:

~>[>]:
file
zero or more lines to be written to *file*
~>

Examples

A sample command line might be:

```
cu -s2400 -l/dev/cua0 4479801
```

Where "-s2400" indicates a line speed of 2400 baud, and "-l/dev/cua0" supplies the device name of the communications line. The system looks in the file L-devices for a line with these characteristics."4479801" is the phone number of the remote system. For a directly connected line:

```
cu -l/dev/tty01 dir
```

"dir" indicates a direct line connection.

To dial out of an internal phone system, such as a computerized branch exchange (CBX):

```
cu -s2400 -l/dev/cua0 9,4479801
```

If your system does not automatically wait for an outgoing dial tone, use several commas for an extended delay:

```
cu -s2400 -l/dev/cua0 9,,,4479801
```

Files

/usr/lib/uucp/L-devices	Device information
/usr/spool/uucp/LCK..(tty-device)	Lockout mechanism
/usr/lib/uucp/dial	Dialer program
/dev/null	

See Also

cat(C), echo(C), stty(C), tty(M)

Diagnostics

Exit code is zero for normal exit, nonzero otherwise.

Comments

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

ASCII files only can be transferred using **~%take** or **~%put** ; binary files cannot be transferred.

Command Reference

IBM Personal Computer XENIX™ Operating System

Programming Family

Personal Computer Software

First Edition (December 1984)

IBM Personal Computer XENIX Library Overview

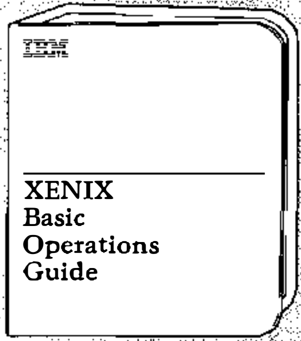
The XENIX[1] System has three available products. They are the:

- Operating System
- Software Development System
- Text Formatting System

The following pages outline the XENIX Operating System library.

[1] XENIX is a trademark of Microsoft Corporation.

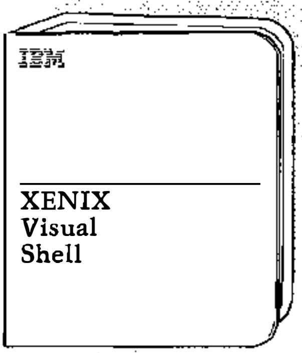
XENIX Operating System Library



For the New User

- Basic concepts
- Logging in and out
- Preparing your terminal
- Editing with text editors
- Introduction to the Bourne Shell
- Sending and receiving network mail and messages

An introduction to the XENIX system, outlining the fundamental concepts and procedures you need to operate the system.

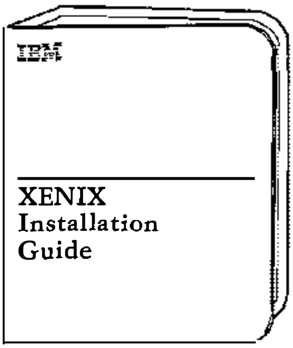


For the New User

- Special keys and definitions
- Command menus
- Creating and managing files and directories
- Customized commands and menus
- Help information

A description of the Visual Shell, the interface between you and the Operating System. Offers hands-on examples.

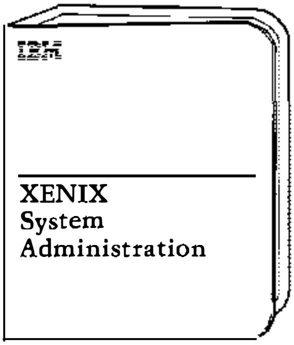
For the System Installer



- Fixed disk preparation
- Operating, Software Development, and Text Formatting System installation
- Creating a Super-User password
- Co-residence with other operating systems

A guide to the installation and management of the XENIX system on your computer.

For the System Administrator



- Starting and stopping the system
- Using, maintaining and backing up files
- Solving system problems
- Using peripheral devices
- Adding users to the system

A guide to managing and maintaining the system.

For All Users

- Alphabetic command listing
- Command definition
- Command syntax
- Command usage
- System commands, functions, and files

A comprehensive command reference, including a concise and complete description of (C) commands, (M) (F) system commands, functions and files.

About This Book

This manual is a reference for programmers who use or intend to use the XENIX Operating System. This manual provides a listing of all the available commands in the (C), (M), and (F) sections of the XENIX Operating System. The (C) stands for Command, (M) stands for Miscellaneous, and (F) stands for File Format section.

In this book are the names, syntax, descriptions, and examples. Comments are also included when necessary. The (M) section contains miscellaneous information, including a great deal of system maintenance information. The (F) section outlines the formats of various files.

Related XENIX Publications

- IBM Personal Computer *XENIX Installation Guide*
- IBM Personal Computer *XENIX Visual Shell*
- IBM Personal Computer *XENIX System Administration*
- IBM Personal Computer *XENIX Basic Operations Guide*

Contents

DATE(C)

Name

date - Prints and sets the date.

Syntax

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

Description

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

```
date 10080045
```

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

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

Field Descriptors:

n	Inserts a newline character
t	Inserts a tab character
m	Month of year - 01 to 12
d	Day of month - 01 to 31
y	Last two digits of year - 00 to 99
D	Date as mm/dd/yy
H	Hour - 00 to 23
M	Minute - 00 to 59
S	Second - 00 to 59
T	Time as hh:mm:ss
j	Julian date - 001 to 366
w	Day of the week - Sunday = 0
a	Abbreviated weekday - Sun to Sat
h	Abbreviated month - Jan to Dec
r	Time in A.M./P.M. notation

Example

The line:

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

generates as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

Diagnostics

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

DC(C)

Name

dc - Invokes an arbitrary precision calculator.

Syntax

dc [*file*]

Description

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

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

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

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

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

d	The top value on the stack is duplicated.
p	The top value on the stack is printed. The top value remains unchanged. **p** interprets the top of the stack as an ASCII string, removes it, and prints it.
f	All values on the stack are printed.
q	Exits the program. If a string is being executed, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped, and the string execution level is popped by that value.
x	Treats the top element of the stack as a character string and executes it as a string of **dc** commands.
X	Replaces the number on the top of the stack with its scale factor.
[. . .]	Puts the bracketed ASCII string onto the top of the stack.
<*x* >*x* =*x*	The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.
v	Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.
!	Interprets the rest of the line as a XENIX 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.

z The stack level is pushed onto the stack.

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

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

;: Used by **bc** for array operations.

Example

This example prints the first ten values of n!:

```
[lal+dsa*plal0>y]sy0sal
lyx
```

See Also

bc(C)

Diagnostics

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

Comments

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

DD(C)

Name

dd - Converts and copies a file.

Syntax

```
dd [option=value] ...
```

Description

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

Option	*Value*
if=*file*	Input filename; standard input is default.
of=*file*	Output filename; standard output is default.
ibs=*n*	Input block size *n* bytes (default, 512).
obs=*n*	Output block size (default, 512).
bs=*n*	Sets both input and output block size, superseding **ibs** and **obs;** also, if no conversion is specified, it is particularly efficient because no in-core copy needs to be done.

cbs=*n*	Conversion buffer size.
skip=*n*	Skips *n* input records before starting copy.
seek=*n*	Seeks *n* records from beginning of output file before copying.
count=*n*	Copies only *n* input records.
conv=ascii	Converts EBCDIC to ASCII.
conv=ebcdic	Converts ASCII to EBCDIC.
conv=ibm	Slightly different map of ASCII to EBCDIC.
conv=lcase	Maps alphabetics to lowercase.
conv=ucase	Maps alphabetics to uppercase.
conv=swab	Swaps every pair of bytes.
conv=noerror	Does not stop processing on an error.
conv=sync	Pads every input record to *ibs.*
conv=" . . . , . . . "	Several comma-separated conversions.

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

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

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

Example

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

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

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

See Also

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

Diagnostics

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

Comments

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

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

DEVNM(C)

Name

devnm - Identifies device name.

Syntax

/etc/devnm [*names*]

Description

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

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

Examples

Be sure to type full path names in this example:

```
/etc/devnm /usr
```

If **/dev/**hdØ3 is mounted on **/usr**, this produces:

```
hdØ3 /usr
```

Files

/dev/*	Device names
/etc/rc	XENIX startup commands

See Also

setmnt(C)

DF(C)

Name

df - Reports the number of free disk blocks.

Syntax

```
df [ -t ] [ -f ] [ filesystem . . . ]
```

Description

The **df** command prints out the number of free blocks and free inodes available for on-line file systems by examining the counts kept in the super-blocks. One or more *filesystem* arguments may be specified by device name (for example, **/dev/hd03** or **/dev/usr**). If the *filesystem* argument is unspecified, the free space on all mounted file systems is sent to the standard output. The list of mounted file systems is given in **/etc/mnttab**.

The **-t** flag causes the total allocated block figures to be reported as well.

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

Files

/dev/*
/etc/mnttab

See Also

fsck(C), mnttab(F)

Comment

See “Comments” under **mount**(C).

DIFF(C)

Name

diff - Compares two text files.

Syntax

diff [**-efbh**] *file1 file2*

Description

The **diff** command tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is **-**, the standard input is used. If *file1* (*file2*) is a directory, a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 **a** *n3*,*n4*
n1,*n2* **d** *n3*
n1,*n2* **c** *n3*,*n4*

These lines resemble **ed** commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward, one may ascertain equally how to convert *file2* into *file1*. As in **ed**, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

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

The **-e** option produces a script of *a*, *c*, and *d* commands for the editor **ed**, which will recreate *file2* from *file1*. The **-f** option produces a similar script, not useful with **ed**, in the opposite order. In connection with **-e**, the following shell procedure helps maintain multiple versions of a file:

```
(shift; cat $*; echo '1,$p') |ed - $1
```

This works by performing a set of editing operations on an original ancestral file. This is done by combining the sequence of **ed** scripts given as all command line arguments except the first. These scripts are presumed to have been created with **diff** in the order given on the command line. The set ●f editing operations is then piped as an editing script to **ed** where all editing operations are performed on the ancestral file given as the first argument on the command line. The final version of the file is then printed on the standard output. Only an ancestral file ($1) and a chain of version-to-version **ed** scripts ($2,$3, . . .) made by **diff** need be on hand.

Except in rare circumstances, **diff** finds the smallest sufficient set of file differences.

The **-h** option does a faster, less-rigorous job than the default or the **-e**, **-f**, and **-b** options. It works only when changed stretches are short and well separated, but also works on files of unlimited length. The **-e** and **-f** cannot be used with the **-h** option.

Files

/tmp/d?????
/usr/lib/diffh for **-h**

See Also

cmp(C), comm(C), ed(C)

Diagnostics

Exit status is 0 for no differences, 1 for some differences, 2 for errors.

Comment

Editing scripts produced under the **-e** or **-f** option do not always work correctly on lines consisting of a single period (.).

DIFF3(C)

Name

diff3 - Compares three files.

Syntax

```
diff3 [ -ex3 ] file1 file2 file3
```

Description

The **diff3** command 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 performed in converting a given range of a given file to some other range is indicated in one of these ways:

f : n1 a	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f : n1 , n2 c	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file are suppressed.

Under the **-e** option, **diff3** publishes a script for the editor **ed** that will incorporate into *file1* all changes between *file2* and *file3*, that is, the changes that normally would be flagged ==== and ====3. The **-x** option produces a script to incorporate changes flagged with "====". Similarly, the **-3** option produces a script to incorporate changes flagged with "====3". The following command applies a resulting editing script to *file1*:

```
(cat script; echo '1,$p') | ed - file1
```

Files

/tmp/d3*
/usr/lib/diff3prog

See Also

diff(C)

Comments

The **-e** option does not work properly for lines consisting of a single period.

The input file size limit is 64K bytes.

DIRCMP(C)

Name

dircmp - Compares directories.

Syntax

```
dircmp [ -d ] [ -s ] dir1 dir2
```

Description

The **dircmp** command examines *dir1* and *dir2* and generates tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated in addition to a list that indicates whether the files common to both directories have the same contents.

Two options are available:

-d Performs a full **diff** on each pair of like-named files if the contents of the files are not identical

-s Reports whether files are "same" or "different"

See Also

cmp(C), diff(C)

DIRNAME(C)

Name

dirname - Delivers directory part of path name.

Syntax

```
dirname string
```

Description

The **dirname** command delivers all but the last component of the path name in *string* and prints the result on the standard output. If there is only one component in the path name, only a "dot" is printed. It is normally used inside substitution marks (\` \`) within shell procedures.

The companion command **basename** deletes any prefix ending in a slash (/) and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output.

Examples

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

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

This example prints **/a/b/c** on the standard output:

```
dirname /a/b/c/d
```

This example prints a "dot" on the standard output:

```
dirname file.ext
```

See Also

basename(C), sh(C)

DISABLE(C)

Name

disable - Turns off terminals.

Syntax

disable [**-d**] [**-e**] *tty* . . .

Description

This program manipulates the **/etc/ttys** file and signals **init** to disallow logins on a particular terminal. The **-d** and **-e** options "disable" and "enable" terminals, respectively.

Examples

A simple example follows:

```
disable tty01
```

Multiple terminals can be disabled or enabled using the **-d** and **-e** switches before the appropriate terminal name:

```
disable tty01 -e tty02 -d tty03 tty04
```

> **Warning:** Be absolutely certain to pause at least one minute before reusing this command or before using the **enable** command. Failure to do so may cause the system to go down.

Files

/dev/tty*
/etc/ttys

See Also

login(M), enable(C), ttys(M), getty(M), init(M)

DOS(C)

Name

dos - Accesses DOS files.

Syntax

doscat [**-r**] *file*

doscp [**-r**] *file1 file2*

doscp [**-r**] *file* ... *directory*

dosdir *directory*

dosls *directory*

dosmkdir *directory*

dosrm *file*

dosrmdir *directory*

Description

The **dos** commands provide access to the files and directories on DOS disks. The commands perform the following actions:

doscat	Copies one or more DOS files to the standard output. If **-r** is given, the files are copied without newline conversions (see "Conversions" below).
doscp	Copies files between an DOS disk and a XENIX file system. If *file1* and *file2* are given, *file1* is copied to *file2*. If a *directory* is given, one or more *files* are copied to that directory. If the **-r** is given, the files are copied without newline conversions (see "Conversions" below).

dosdir	Lists DOS files in the standard DOS style directory format.
dosls	Lists DOS directories and files in a XENIX style (see **ls**(C)).
dosrm	Removes files from a DOS disk.
dosmkdir	Creates a directory on a DOS disk.
dosrmdir	Deletes directories from a DOS disk.

The *file* and *directory* arguments for DOS files and directories have the form:

```
device:name
```

where *device* is a XENIX pathname for the special device file containing the DOS disk, and *name* is a pathname to a file or directory on the DOS disk. The two components are separated by a colon (**:**). For example, the argument:

```
/dev/fdØ:/src/file.asm
```

specifies the DOS file, **file.asm**, in the directory, **/src**, on the disk in the device file **/dev/fdØ**. Note that slashes (and not backslashes) are used as filename separators for DOS pathnames. Arguments without a *device* : are assumed to be XENIX files.

For convenience, the drive letters A: and B: can be used for any 48TPI DOS diskette in drives 0 and 1 respectively. The drive letters X: and Y: can be used for any 96TPI DOS diskette in drives 0 and 1 respectively. You must enter the drive letter as a capital letter when you use this notation to name a drive.

The commands operate on the following kinds of disks:

5 1/4 inch DOS
8 or 9 sectors per track
40 tracks per side
1 or 2 sides
DOS version 1 or ≥2.

Converisons

All DOS text files use a carriage-return/linefeed combination, CR-LF, to indicate a newline. XENIX uses a single newline LF character. When the **doscat** and **doscp** commands transfer DOS text files to XENIX, they automatically strip the CR. When text files are transferred to DOS, the commands insert the CR before each LF character. The **-r** option can be used to override the automatic conversion and force the command to perform a true byte copy regardless of file type.

Examples

```
doscat /dev/fd096ds15:/docs/memo.txt
or doscat x:/docs/memo.txt

dosdir /dev/fd048ds9 or dosdir A:

doscp /tmp/myfile.txt /dev/fd148ds9:/docs/memo.txt
or doscp /tmp/myfile.txt B:/docs/memo.txt

dosls /dev/fd0:/src or dosls X:/src

dosmkdir /dev/fd0:/usr/docs or dosmkdir X:/usr/docs

dosrm /dev/fd196ds15:/docs/memo.txt
or dosrm Y:/docs/memo.txt

dosrmdir /dev/fd0:/usr/docs or dosrmdir X:/usr/docs
```

Files

/etc/default/msdos	Default information
/dev/fd*	Diskette drive devices

See Also

dtype(C), default(M)

Comment

It is not possible to refer to DOS directories with wild card specifications. It is the user's responsibility to ensure he has exclusive access to the device containing the DOS disk. If two or more processes simultaneously attempt to access the DOS disk the result is unpredictable.

DTYPE(C)

Name

dtype - Determines disk type.

Syntax

```
dtype [-s] device ...
```

Description

The **dtype** command determines type of disk, prints pertinent information on the standard output unless the silent (**-s**) option is selected, and exits with a corresponding code (see below). When more than one argument is given, the exit code corresponds to the last argument.

Disk Type	Exit Code	Message (optional)
Misc.	60	error (specified)
	61	empty or unrecognized data
Storage	70	backup format, volume n
	71	tar format[, extent e of n]
	72	cpio format
	73	cpio character (-c) format
DOS	80	DOS 1.x, 8 sec/track, single sided
	81	DOS 1.x, 8 sec/track, dual sided

Disk Type	Exit Code	Message (optional)
	90	DOS 2.x, 8 sec/track, single sided
	91	DOS 2.x, 8 sec/track, dual sided
	92	DOS 2.x, 9 sec/track, single sided
	93	DOS 2.x, 9 sec/track, dual sided
XENIX	120	XENIX 2.x filesystem[needs cleaning]
	130	XENIX 3.x filesystem[needs cleaning]

Comment

XENIX file systems and dump and cpio binary formats may not be recognized if created on a foreign system. This is due to such system differences as byte and word swapping and structure alignment.

DU(C)

Name

du - Summarizes disk usage.

Syntax

```
du [-afrsu] [names]
```

Description

The **du** command gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, the current directory is used.

The optional argument **-s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **-a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

Normally, **du** is silent about directories that cannot be read, files that cannot be opened, etc. The **-r** option causes **du** to generate messages in such instances.

A file with two or more links is only counted once.

Comments

If the **-a** option is not used, nondirectories given as arguments are not listed.

If there are too many distinct linked files, **du** counts the excess files more than once. Files with holes in them get an incorrect block count.

DUMPDIR(C)

Name

dumpdir - Prints the names of files on a backup archive.

Syntax

dumpdir [f *filename*]

Description

The **dumpdir** command is used to list the names and inode numbers of all files and directories on an archive written with the **backup** command. This is most useful when attempting to determine the location of a particular file in a set of backup archives.

The **f** option causes *filename* to be used as the name of the backup device instead of the default. The backup device depends on the setting of the variable TAPE in the file **/etc/default/backup** .

File

rst* Temporary files

See Also

backup(C), restor(C), default(M)

Diagnostics

If the backup extends over more than one volume (where a volume is likely a diskette or tape), you will be asked to change volumes. Press Enter after changing volumes.

ECHO(C)

Name

echo - Echoes arguments.

Syntax

```
echo [-n][-e][-u][--][arg] . . .
```

Description

The **echo** command writes its arguments separated by blanks and terminated by a newline on the standard output. The following options are recognized:

-n	Prints line without a newline.
-e	Prints arguments on the standard error output.
-u	Uses unbuffered I/O when printing.
--	Prints *arg* exactly so that an argument beginning with a dash (for example, **-e** or **-n**) can be specified.

The **echo** command also understands C-like escape conventions. The following escape sequences need to be quoted so that the shell interprets them correctly:

\b	Backspace
\c	Prints line without newline; same as use of **-n** option.
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\	Backslash
\ *n*	The 8-bit character whose ASCII code is the one-, two-, or three-digit octal number *n*, which must start with a zero.

The **echo** command is useful for producing diagnostics in command files and for sending known data into a pipe.

See Also

sh(C)

Comment

Thc **-e** option is a XENIX-specific enhancement and may not be present in other UNIX implementations. Therefore, the application developer should consider the impact to portability when using this feature.

ED(C)

Name

ed - Invokes the text editor.

Syntax

ed [-] [*file*]

Description

The **ed** command invokes the standard text editor, **ed**. If the *file* argument is given, **ed** simulates an **e** command (see below) on the named file; that is to say, the file is read into **ed**'s buffer so that it can be edited. The optional **-** suppresses the printing of character counts by **e**, **r**, and **w** commands, of diagnostics from **e** and **q** commands, and of the ! prompt after a ! shell command. The editor operates on a copy of the file being edited; changes made to the copy have no effect on the file until a **w** (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Commands to **ed** have a simple and regular structure: zero, one, or two addresses followed by a single-character command, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While **ed** is accepting text, it is said to be in input mode. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

The editor supports a limited form of regular expression notation; regular expressions are used in addresses to specify lines and in some commands (for example, **s**) to specify portions of a line that are to be substituted. A regular expression specifies a set of character strings. A member of this set of strings is said to be matched by the regular expression. The regular expressions allowed by **ed** are constructed as follows:

The following one-character regular expressions match a single character:

1.1 An ordinary character (not one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash (\ ) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

a. **. * [** and \ (dot, asterisk, left bracket, and backslash, respectively), which are always special, except when they appear within brackets ([ ]; see 1.4 below).

b. ^ (caret), which is special at the beginning of an entire regular expression (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of brackets ([ ]) (see 1.4 below).

c. **$** (dollar sign), which is special at the end of an entire regular expression (see 3.2 below).

d. The character used to bound (that is, delimit) an entire regular expression, which is special for that regular expression (for example, see how slash (/) is used in the **g** command, below).

1.3 A period (.) is a one-character regular expression that matches any character except newline.

1.4 A nonempty string of characters enclosed in brackets ([ ]) is a one-character regular expression that matches any one character in that string. If, however, the first character of the string is a caret (^), the one-character regular expression matches any character except newline and the remaining characters in the string. The asterisk (*) has this special meaning only if it occurs first in the string. The hyphen (-) may be used to indicate a range of consecutive ASCII characters; for example, [0-9] is equivalent to [0123456789]. The hyphen (-) loses this special meaning if it occurs first (after an initial caret (^), if any) or last in the string. The right bracket (]) does not terminate such a string when it is the first character within it (after an initial caret (^), if any); for example, []**a-f**] matches either a right bracket (]) or one of the letters "a" through "f" inclusive. Period, asterisk, left bracket, and the backslash lose their special meaning within such a string of characters.

The following rules may be used to construct regular expressions from one-character regular expressions:

2.1 A one-character regular expression matches whatever the one-character regular expression matches.

2.2 A one-character regular expression followed by an asterisk (*) is a regular expression that matches zero or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character regular expression followed by \{ *m*\}, \{ *m*,\}, or \{ *m*,*n*\} is a regular expression that matches a range of occurrences of the one-character regular expression. The values of *m* and *n* must be non-negative integers less than 256; \{ *m*\} matches exactly *m* occurrences; \{ *m*,\} matches at least *m* occurrences; \{ *m*,*n*\} matches any number of occurrences between *m* and *n*, inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

2.4 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.5 A regular expression enclosed between the character sequences **\\(** and **\\)** is a regular expression that matches whatever the unadorned regular expression matches. See 2.6 below for a discussion of why this is useful.

2.6 The expression \ *n* matches the same string of characters as was matched by an expression enclosed between **\\(** and **\\)** earlier in the same regular expression. Here *n* is a digit; the sub-expression specified is that beginning with the *n -th* occurrence of **\\(** counting from the left. For example, the expression **∧\\(.*\\)\\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an entire regular expression may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A caret (**∧**) at the beginning of an entire regular expression constrains that regular expression to match an initial segment of a line.
3.2 A dollar sign (**$**) at the end of an entire regular expression constrains that regular expression to match a final segment of a line. The construction **∧** *entire regular expression***$** constrains the entire regular expression to match the entire line.

The null regular expression (that is, **//**) is equivalent to the last regular expression encountered.

To understand addressing in **ed** it is necessary to know that there is a current line at all times. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. Addresses are constructed as follows:

1. The character **.** addresses the current line.

2. The character **$** addresses the last line of the buffer.

3. A decimal number *n* addresses the *n -th* line of the buffer.

4. *x* addresses the line marked with the mark name character *x*, which must be a lowercase letter. Lines are marked with the **k** command described below.

5. A regular expression enclosed by slashes (/) addresses the first line found by searching forward from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched.

6. A regular expression enclosed in question marks (?) addresses the first line found by searching backward from the line preceding the current line toward the beginning of the buffer and stopping at the first line containing a string matching the regular expression. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before "Files " below.

7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus or minus the indicated number of lines. The plus sign may be omitted.

8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; for example, **-5** is understood to mean **.-5**.

9. If an address ends with + or -, 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -). Moreover, trailing + and - characters have a cumulative effect, so - - refers to the current line less 2.

10. For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last addresses are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5 and 6). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per filename, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory.

When reading a file, **ed** discards ASCII Nul characters and all characters after the last newline. Files that contain characters not in the ASCII set (bit 8 on) cannot be edited by **ed**.

If the closing delimiter of a regular expression or of a replacement string, for example, (/), would be the last character before a newline, that delimiter may be omitted, in which case the addressed line is printed. Thus the following pairs of commands are equivalent:

```
s/s1/s2
s/s1/s2/p

g/s1
g/s1/p

?s1
?s1?
```

In the following list of **ed** commands, the default addresses are shown in parentheses. The parentheses are not part of the address; they show that the given addresses are the default.

In general, only one command may appear on a line. However, any command (except **e**, **f**, **r**, or **w**) may be suffixed by **p** or by **l**, in which case the current line is either printed or listed, respectively, as discussed below under the **p** and **l** commands.

(.)a
<text>
.

The **a**ppend command reads the given text and appends it after the addressed line; dot is left at the last inserted line, or, if there were no inserted lines, at the addressed line. Address 0 is legal for this command: it causes the appended text to be placed at the beginning of the buffer.

(.)c
<text>
.

The **c**hange command deletes the addressed lines, then accepts input text that replaces these lines; dot is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

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

e *file*

The **e**dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; dot is set to the last line of the buffer. If no filename is given, the currently remembered filename, if any, is used (see the **f** command). The number of characters read is typed; *file* is remembered for possible use as a default filename in subsequent **e**, **r**, and **w** commands. If *file* begins with an exclamation (**!**), the rest of the line is taken to be a shell command. The output of this command is read for the **e** and **r** commands. For the **w** command, the file is used as the standard input for the specified command. Such a shell command is not remembered as the current filename.

E: *file*

The **E**dit command is like **e**, except the editor does not check to see if any changes have been made to the buffer since the last **w** command.

f *file*

If *file* is given, the **f**ilename command changes the currently remembered filename to *file*; otherwise, it prints the currently remembered filename.

(1,$)g/*regular-expression*/*command list*

In the **g**lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, the given command list is executed with **.** initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \ ; **a**, **i**, and **c** commands and associated input are permitted; the **.** terminating input mode may be omitted if it would be the last line of the command list. An empty command list is equivalent to the **p** command. The **g**, **G**, **v**, and **V** commands are not permitted in the command list. See also "Comments " and the last paragraph before "Files " below.

(1,$)G/*regular-expression***/**

In the interactive **G**lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, that line is printed, dot (**.**) is changed to that line, and any one command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a newline acts as a null command; an ampersand (**&**) causes the re-execution of the most recent command executed within the current invocation of **G**. The commands input as part of the execution of the **G** command may address and affect any lines in the buffer. The **G** command can be terminated by typing an Interrupt (Del).

h

The **h**elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The **H**elp command causes **ed** to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It also explains the previous diagnostic if there was one. The **H** command alternately turns this mode on and off; it is initially on.

(.)i
<text>
.

The **i**nsert command inserts the given text before the addressed line; dot is left at the last inserted line, or if there were no inserted lines, at the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 is not legal for this command.

(.,.+1)j

The **j**oin command joins contiguous lines by removing the appropriate newline characters. If only one address is given, this command does nothing.

(.)k*x*

The mark command marks the addressed line with name *x*, which must be a lowercase letter. The address '*x* then addresses this line; dot is unchanged.

(.,.)l

The list command prints the addressed lines in an unambiguous way: a few nonprinting characters (for example tab, backspace) are represented by mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An **l** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)m *a*

The move command repositions the addressed lines after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed lines to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; dot is left at the last line moved.

(.,.)n

The number command prints the addressed lines, preceding each line by its line number and a tab character; dot is left at the last line printed. The **n** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)p

The print command prints the addressed lines; dot is left at the last line printed. The **p** command may be appended to any command other than **e**, **f**, **r**, or **w**; for example, **dp** deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The **P** command alternately turns this mode on and off; it is initially on.

q

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

Q

The editor exits without checking if changes have been made in the buffer since the last **w** command.

($)r *file*

The **r**ead command reads in the given file after the addressed line. If no filename is given, the currently remembered filename, if any, is used (see **e** and **f** commands). The currently remembered filename is not changed unless *file* is the very first filename mentioned since **ed** was invoked. Address 0 is legal for **r** and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; dot is set to the last line read in. If *file* begins with !, the rest of the line is taken to be a shell (**sh**(C)) command whose output is to be read. Such a shell command is not remembered as the current filename.

(.,.)s/*regular-expression***/***replacement***/**
  or
(.,.)s/*regular-expression***/***replacement***/g**

The **s**ubstitute command searches each addressed line for an occurrence of the specified regular expression. In each line in which a match is found, all non-overlapped matched strings are replaced by the replacement if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or newline may be used instead of / to delimit the regular expression and the replacement; dot is left at the last line on which a substitution occurred.

An ampersand (**&**) appearing in the replacement is replaced by the string matching the regular expression on the current line. The special meaning of the ampersand in this context may be suppressed by preceding it with a backslash. The characters \ *n*, where *n* is a digit, are replaced by the text matched by the *n -th* regular subexpression of the specified regular expression enclosed between \(and \) . When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the replacement, the replacement used in the most recent substitute command is used as the replacement in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a backslash (\ ).

A line may be split by substituting a newline character into it. The newline in the replacement must be escaped by preceding it with a backslash (\ ). Such a substitution cannot be done as part of a **g** or **v** command list.

(.,.)t*a*

This command acts just like the **m** command, except that a copy of the addressed lines is placed after address *a* (which may be 0); dot is left at the last line of the copy.

u

The **u**ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, **G**, or **V** command.

(1,$)v/*regular-expression/command list*

This command is the same as the global command **g** except that the command list is executed with dot initially set to every line that does not match the regular expression.

(1,$)V/*regular-expression/*

This command is the same as the interactive global command **G** except that the lines that are marked during the first step are those that do not match the regular expression.

(1,$)w *file*

The **write** command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writeable by everyone), unless the umask setting (see **sh**(C)) dictates otherwise. The currently remembered filename is not changed unless *file* is the very first filename mentioned since **ed** was invoked. If no filename is given, the currently remembered filename, if any, is used (see **e** and **f** commands); dot is unchanged. If the command is successful, the number of characters written is displayed. If *file* begins with an exclamation (!), the rest of the line is taken to be a shell command to which the addressed lines are supplied as the standard input. Such a shell command is not remembered as the current filename.

X

A key string is demanded from the standard input. Subsequent **e**, **r**, and **w** commands will encrypt and decrypt the text with this key by the algorithm of **crypt**(C). An explicitly empty key turns off encryption.

($)=

The line number of the addressed line is typed; dot is unchanged by this command.

! *shell command*

The remainder of the line after the ! is sent to the XENIX shell (**sh**(C)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered filename; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; dot is unchanged.

(.+1)

An address alone on a line causes the addressed line to be printed. An Enter alone on a line is equivalent to **.+1p**. This is useful for stepping forward through the editing buffer a line at a time.

If an interrupt signal (ASCII Del or Break) is sent, **ed** prints a question mark (**?**) and returns to its command level.

Files

/tmp/e#	Temporary; # is the process number
ed.hup	Work is saved here if the terminal stops.

See Also

grep(C), sed(C), sh(C)

Diagnostics

?	Command errors
? *file*	An inaccessible file

p. Use the **h**elp and **H**elp commands for detailed explanations.

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, **ed** warns the user if an attempt is made to destroy **ed**'s buffer via the **e** or **q** commands: it prints **?** and allows you to continue editing. The hyphen (-) command-line option inhibits this feature. A second **e** or **q** command at this point will take effect.

Comments

An exclamation (!) command cannot be subject to a **g** or a **v** command. The **!** command and the **!** escape from the **e**, **r**, and **w** commands cannot be used if the the editor is invoked from a restricted shell (see **sh**(C)).

The sequence **\n** in a regular expression does not match any character.

The **l** command mishandles Del.

Because 0 is an illegal address for the **w** command, it is not possible to create an empty file with **ed**.

ENABLE(C)

Name

enable - Turns on terminals.

Syntax

enable [**-d**][**-e**] *tty*...

Description

This program manipulates the **/etc/ttys** file and signals **init** to allow logins on a particular terminal. The **-e** and **-d** options may be used to allow logins on some terminals and disallow logins on other terminals in a single command.

> **Warning:** Be absolutely certain to pause at least one minute before reusing this command or before using the **disable** command. Failure to do so may cause the system to go down.

Examples

A simple command to enable **tty**01 follows:

```
enable tty01
```

Multiple terminals can be disabled or enabled using the **-d** and **-e** switches before the appropriate terminal name:

```
enable tty01 -e tty02 -d tty03 tty04
```

Files

/dev/tty*
/etc/ttys

See Also

login(M), disable(C), ttys(M), getty(M), init(M)

ENV(C)

Name

env - Sets environment for command execution.

Syntax

env [-][*name=value*] . . . [*command args*]

Description

The **env** command obtains the current environment, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name* = *value* are merged into the inherited environment before the command is executed. The **-** flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

See Also

sh(C), profile(M), environ(M)

EX(C)

Name

ex - Invokes a text editor.

Syntax

```
ex [-] [-v] [-t tag] [-r] [+lineno] name ...
```

Description

The **ex** editor is the root of the editors **ex** and **vi**. The **ex** editor is a superset of **ed**, whose most notable extension is a display editing facility. Display based editing is the focus of **vi**.

If you have a CRT terminal, you may wish to use a display based editor; in this case, see **vi**(C), a command that focuses on the display editing portion of **ex.**

For ed Users

If you have used **ed** you will find that **ex** has a number of new features. Generally, the **ex** editor uses far more of the capabilities of terminals than **ed** does. It uses the terminal capability database **termcap(M)** 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 **ex** editor makes use of features such as insert and delete character and line in its visual command mode, which can be abbreviated **vi**, which is the central mode of editing when using **vi**(C). There is also an interline editing **open** command, (**o**) that works on all terminals.

The **ex** editor contains a number of features for easily viewing the text of a file. The **z** command gives easy access to windows of text. Pressing Ctrl-D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just pressing the Enter key. Of course, the screen-oriented visual mode gives constant access to editing context.

The **ex** editor gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change. This editor also gives you feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents the overwriting of existing files unless you have edited them, so that you don't accidentally overwrite with a **write**, a file other than the one you are editing. If the system (or editor) goes down, or you accidentally hang up the phone, you can use the **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Several features in **ex** permit editing 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 edit each in turn. You can also give the **next** command a list of filenames, or a pattern used by the shell to specify a new set of files to be edited. 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 you want to find the definition of a particular function in a large program.

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.

The command **&** repeats the last **substitute** command. There is also 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 uppercase and lowercase. It is possible to ignore case in searches and substitutions. The **ex** editor also allows regular expressions that match words to be constructed. This is convenient, for example, when searching for the word "edit" if your document also contains the word "editor."

In **ex**, you can set options. One option that is very useful is the **autoindent** option that allows the editor to automatically supply leading white space to align text. You can then use the Ctrl-D key to backtab, space, and tab forward to align new code easily.

Miscellaneous useful features include an intelligent **join** (j) command that supplies whitespace between joined lines automatically, the commands < and > that shift groups of lines, and the ability to filter portions of the buffer through commands such as **sort**.

Files

/usr/lib/ex3.7strings	Error messages
/usr/lib/ex3.7recover	Recover command
/usr/lib/ex3.7preserve	Preserve command
/etc/termcap	Terminal capability
$HOME/.exrc	Editor startup file
/tmp/Ex*nnnnn*	Editor temporary file
/tmp/Rx*nnnnn*	Named buffer temporary file
/usr/preserve	Preservation directory

See Also

awk(C), ed(C), grep(C), sed(C), termcap(M), vi(C)

Credit

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

Comments

The **undo** command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

The **undo** command 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.

EXPR(C)

Name

expr - Evaluates arguments as an expression.

Syntax

expr *arguments*

Description

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that 0 is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Expressions should be quoted by the shell, because many characters that have special meaning in the shell also have special meaning in **expr**. The list is in order of increasing precedence, with equal precedence operators grouped within braces ({ and }).

expr* | *expr
: Returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr* & *expr
: Returns the first **expr** if neither **expr** is null nor **0**, otherwise returns **0**.

***expr*{=,>, >=,<, <=, !=+} expr**
: Returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, - } *expr*
Addition or subtraction of integer-valued arguments.

expr { *,/,% } *expr*
Multiplication, division, or remainder of integer-valued arguments.

expr : *expr*
The matching operator : compares the first argument with the second argument, which must be a regular expression; regular expression syntax is the same as that of **ed**(C), except that all patterns are "anchored" (That is, begin with a caret (∧)). and therefore the caret is not a special character in that context. (Note that in the shell, the caret has the same meaning as the pipe symbol (|).) Normally the matching operator returns the number of characters matched (zero on failure). Alternatively, the \(. . . \) pattern symbols can be used to return a portion of the first argument.

Examples

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

Adds 1 to the shell variable **a**.

```
2.   For $a equal to either "/usr/abc/file" or just "/file"
     'expr $a : '.*/\(.*\)'| $a'
```

Returns the last segment of a path name (for example, **file**). Watch out for the slash alone as an argument: **expr** will take it as the division operator (see "Comments" below).

```
3.   expr $VAR : '.*'
```

Returns the number of characters in **$VAR**.

See Also

ed(C), sh(C)

Diagnostics

As a side effect of expression evaluation, **expr** returns the following exit values:

0	If the expression is neither null nor zero.
1	If the expression is null or zero.
2	For invalid expressions.

Other diagnostics include:

syntax error	For operator/operand errors
nonnumeric argument	If arithmetic is attempted on such a string

Comments

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an equal sign (=), the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

Thus the arguments are passed to **expr** (and will all be taken as the = operator). The following permits comparing equal signs:

```
expr X$a = X=
```

FACTOR(C)

Name

factor - Factor a number.

Syntax

```
factor [number]
```

Description

When **factor** is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about $7.2x10^{16}$) it will factor the number and print it; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any nonnumeric character.

If **factor** is invoked with an argument, it factors the number as above and then exits.

The time it takes to factor a number, *n*, is proportional to $\sqrt{n}$.

Diagnostics

Factor returns an error message if the supplied input value is greater than 2^{56} or is not an integer number.

FALSE(C)

Name

false - Returns with a nonzero exit value.

Syntax

false

Description

The **false** command does nothing except return with a nonzero exit value; **true(C)**, **false**'s counterpart, does nothing except return with a zero exit value. False is typically used in shell procedures such as:

```
until false
do
command
done
```

See Also

sh(C), true(C)

Diagnostics

The **false** command has exit status 1.

FILE(C)

Name

file - Determines file type.

Syntax

file [**-m**] *file* . . .

file [**-m**] **-f** *namesfile*

Description

The **file** command performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, **file** examines the first 512 bytes and tries to guess its language.

If the **-f** option is given, **file** takes the list of filenames from *namefile*. If the **-m** option is given, **file** sets the access time for the examined file to the current time. Otherwise, the access time remains unchanged.

Several object file formats are recognized. For **a.out** and **x.out** format object files, the relationship of *cc* flags to file classification is **-i** for "separate", **-n** for "pure", and **-s** for not "not stripped".

Credit

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

Comment

File can make mistakes: in particular, it often suggests that command files are C programs. The access time will always be updated for files which you do not own.

FIND(C)

Name

find - Finds files.

Syntax

```
find pathname-list expression
```

Description

The **find** command recursively descends the directory hierarchy for each path name in the *pathname-list* (that is, one or more path names) seeking files that match a Boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, -*n* means less than *n*, and *n* means exactly *n*.

-name *file*	True if *file* matches the current filename. Normal shell argument syntax may be used if escaped (watch out for the left bracket ([), the question mark (?) and the asterisk(*)).
-perm *onum*	True if the file permission flags exactly match the octal number *onum* (see **chmod(C)**). If *onum* is prefixed by a minus sign, more flag bits (017777) become significant and the flags are compared: (flags&onum)==onum
-type *x*	True if the type of the file is *x*, where *x* is **b** for a block special file, **c** for a character special file, **d** for a directory, **p** for a named pipe, **f** for a plain file, or **n** for a semaphore or shared data file.
-links *n*	True if the file has *n* links.

-user *uname*	True if the file belongs to the user *uname.* If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.
-group *gname*	True if the file belongs to the group *gname.* If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.
-size *n*	True if the file is *n* blocks long (512 bytes per block).
-atime *n*	True if the file has been accessed in *n* days.
-mtime *n*	True if the file has been modified in *n* days.
-ctime *n*	True if the file has been changed in *n* days.
-exec *cmd*	True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon (\;). A command argument {} is replaced by the current path name.
-ok *cmd*	Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.
-print	Always true; causes the current pathname to be printed.
-newer *file*	True if the current file has been modified more recently than the argument *file*.
(*expression*)	True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

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

negation	The negation of a primary is specified with the exclamation (!) unary *not* operator.
AND	The AND operation is implied by the juxtaposition of two primaries.
OR	The OR operation is specified with the **-o** operator given between two primaries.

Examples

The following removes all files named **a.out** or ***.o** that have not been accessed for a week:

```
find  / \( -name a.out -o -name '*.o' \)
-atime +7 -exec rm {}\;
```

Files

/etc/passwd
/etc/group

See Also

cpio(C), sh(C), test(C), cpio(F)

FINGER(C)

Name

finger - Finds information about users.

Syntax

finger [**-bfilpqsw**][*login*1 [*login2* . . .]]

Description

By default, **finger** lists the login name, full name, terminal name and write status (as a "*" before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current XENIX user. (Idle time is minutes if it is a single integer, hours and minutes if a colon (:) is present, or days and hours if a **d** is present.)

A longer format also exists and is used by **finger** whenever a list of names is given. (Account names as well as first and last names of users are accepted.) This is a multi-line format; it includes all the information described above as well as the user's home directory and login shell, any plan that the user has placed in the file *.plan* in the home directory, and the project on which the user is working from the file *.project* also in the home directory.

Finger options are:

-b	Briefer long output format of users.
-f	Suppresses the printing of the header line (short format).
-i	Quick list of users with idle times.
-l	Forces long output format.
-p	Suppresses printing of the *.plan* files.
-q	Quick list of users.

-s Forces short output format.

-w Forces narrow format list of specified users.

Files

/etc/utmp	Who file
/etc/passwd	User names, offices, phones, login directories, and shells.
/usr/adm/lastlog	Last login times.
$HOME/.plan	Plans.
$HOME/.project	Projects.

See Also

who(C)

Credit

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

Comments

Only the first line of the *.project* file is printed.

The "office" column of the output will contain any text in the comment field of the user's **/etc/passwd** file entry that immediately follows a comma (,). For example, if the entry is:

```
johnd:eX8HinAk:201:50:John Doe,321:/usr/johnd:/bin/sh
```

the number 321 will appear in the office column.

Idle time is computed as the elapsed time since any activity on the given terminal. This includes previous invocations of **finger** that may have modified the terminal's corresponding device file /dev/tty??.

FSCK(C)

Name

fsck - Checks and repairs file systems.

Syntax

```
/etc/fsck [options][file-system] . . .
```

Description

The **fsck** command audits and interactively repairs inconsistent conditions for XENIX file systems. If the file system is consistent, 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. Note that most corrective actions result in some loss of data. The amount and severity of the loss 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** defaults to the action of the **-n** option.

The following flags are interpreted by **fsck**:

-y Assumes a yes response to all questions asked by **fsck**.

-n Assumes a no response to all questions asked by **fsck**; do not open the file system for writing.

-s*b*:*c* Ignores the actual free list and (unconditionally) reconstructs a new one by rewriting the super-block of the file system. The file system *must* be unmounted while this is done.

The -s*b*:*c* option allows for creating an optimal free-list organization.

The following forms are supported:

- **-s**
- **-s**Blocks-per-cylinder:Blocks-to-skip (for anything else)

If *b:c* is not given, the values used when the file system was created are used. If these values were not specified, a reasonable default value is used.

-S Conditionally reconstructs the free list. This option is like **-s***b:c* above except that the free list is rebuilt only if there are no discrepancies discovered in the file system. Using **-S** forces 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** prompts the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when **fsck** completes.

-rr Recovers the root file system. The required *file-system* argument must refer to the root file system, and preferably to the block device (normally **/dev/root**). This switch implies **-y** and overrides **-n**. If any modifications to the file system are required, the system is automatically shut down to insure the integrity of the file system.

-c Causes any supported file system to be converted to the type of the current file system. The user is asked to verify the request for each file system that requires conversion unless the **-y** option is specified. It is recommended that every file system be checked with this option, while unmounted if it is to be used with the current version of XENIX. To update the active root file system, it should be checked with:

```
fsck -c -rr /dev/root
```

The *file systems* are specified, **fsck** reads a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are:

- Blocks claimed by more than one inode or the free list.

- Blocks claimed by an inode or the free list outside the range of the file system.

- Incorrect link counts.

- Size checks:
  Incorrect number of blocks.
  Directory size not 16-byte aligned.

- Bad inode format.

- Blocks not accounted for anywhere.

- Directory checks:
  File pointing to unallocated inode.
  Inode number out of range.

- Super-block checks:
  More than 65536 inodes.
  More blocks for inodes than there are in the file system.

- Bad free block list format.

- Total free block 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 pre-exist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before **fsck** is executed).

File

/etc/checklist

Contains default list of file systems to check.

See Also

checklist(F), filesystem(F)

Diagnostics

The diagnostics produced by **fsck** are intended to be self-explanatory.

Comments

The **fsck** command will not run on a mounted nonraw file system unless the file system is the root file system or unless the **-n** option is specified and no writing out of the file system will take place. If any such attempt is made, a warning is printed and no further processing of the file system is done for the specified device.

Although checking a raw device is almost always faster, there is no way to tell if the file system is mounted. Cleaning a mounted file system will almost certainly result in an inconsistent super-block.

> **Warning:** For a Microsoft XENIX 2.3 file system to be properly supported under XENIX, it is necessary that **fsck** be run on each 2.3 file system to be mounted under the XENIX kernel. For the root file system, "fsck -rr /dev/root" should be run and for all other file systems "fsck /dev/??" on the *unmounted* block device should be used.

GETOPT(C)

Name

getopt - Parses command options.

Syntax

```
set -- `getopt optstring $*`
```

Description

The **getopt** command is used to check and break up options in command lines for parsing by shell procedures. *Optstring* is a string of recognized option letters. If a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by whitespace. The special option **--** is used to delimit the end of the options. The **getopt** command will place **--** in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments ($1 $2 . . .) are reset so that each option is preceded by a dash (-) and in its own shell argument; each option argument is also in its own shell argument.

Example

The following code fragment shows how one can process the arguments for a command that can take the options **a** and **b**, and the option **o**, which requires an argument:

```
set - - Ωgetopt abo: $*Ω
if [$? != Ø ]
then
echo $USAGE
exit 2
fi
for i in $*
do
case $i in
-a | -b)     FLAG=$i; shift;;
-o)                OARG=$2;        shift; shift;;
- -)               shift;  break;;
esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg - - file file
```

See Also

sh(C)

Diagnostics

The **getopt** command prints an error message on the standard error when it encounters an option letter not included in *optstring*.

GREP(C)

Name

grep, egrep, fgrep - Searches a file for a pattern.

Syntax

grep [*options*] *expression* [*files*]

egrep [*options*] [*expression*] [*files*]

fgrep [*options*] [*strings*] [*files*]

Description

Commands of the **grep** family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. The **grep** patterns are limited regular *expressions* in the style of **ed**(C); it uses a compact nondeterministic algorithm. The **egrep** patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. The **fgrep** patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

-v Prints all lines but those matching.

-x Prints only exact matches of an entire line (**fgrep** only).

-c Prints only a count of matching lines.

-l Lists only the names of files with matching lines, separated by newlines.

-h Prevents appending the name of the file with the matching line to the matching line. Used when searching multiple files.

-n Precedes each line by its relative line number in the file.

-b Precedes each line by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

-s Suppresses error messages produced for nonexistent or unreadable files (**grep** only).

-y Turns on matching of letters of either case in the input so that case is insignificant. Does not work for **egrep**.

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

-f *file* The regular *expression* for **grep** or **egrep**, or *strings* list (for **fgrep**) is taken from the *file*.

In all cases, the filename is output if there is more than one input file. Care should be taken when using the characters **$**, *****, **[**, ^ , | , (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotation marks.

The **fgrep** command searches for lines that contain one of the *strings* separated by newlines.

The **egrep** command accepts regular expressions as in **ed**(C), except for \(and \) , with the addition of the following:

- A regular expression followed by a plus sign (+) matches one or more occurrences of the regular expression.
- A regular expression followed by a question mark (?) matches 0 or 1 occurrences of the regular expression.
- Two regular expressions separated by a vertical bar (|) or by a newline match strings that are matched by either regular expression.
- A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is **[]**, then ***?+**, then concatenation, then the backslash (\ ) and the newline.

See Also

ed(C), sed(C), sh(C)

Diagnostics

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

Comments

Ideally there should be only one **grep**, but there isn't a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters; longer lines are truncated.

The **egrep** command does not recognize ranges, such as [**a-z**], in character classes.

When using **grep** with the **-y** option, the search is not made totally case insensitive in character ranges specified within brackets.

Multiple strings can be specified in **fgrep** without using a separate strings file by using the quoting conventions of the shell to imbed newlines in the *single* string argument. For example, you might type the following at the command line:

```
fgrep 'string1
string2
string3' text.file
```

Similarly, multiple strings can be specified in **egrep** by doing:

```
egrep 'string1|string2|string3' text.file
```

Thus **egrep** can do almost anything that **grep** and **fgrep** can do.

GRPCHECK(C)

Name

grpcheck - Checks group file.

Syntax

pwcheck [*file*]

grpcheck [*file*]

Description

The **grpcheck** command verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

Files

/etc/group
/etc/passwd

See Also

pwcheck(C), group(M), passwd(M)

Diagnostics

Group entries in **/etc/group** with no login names are flagged.

HALTSYS(C)

Name

haltsys - Closes out the file systems and halts the system.

Syntax

```
/etc/haltsys
```

Description

The **haltsys** command does a shutdn() system call to flush out pending disk I/O, mark the file systems clean, and halt the processor. The **haltsys** command takes effect immediately, so user processes should be terminated beforehand. The **shutdown**(C) is recommended for normal system termination; it warns the users, performs system house cleaning, and calls **haltsys**. Use **haltsys** directly only if some system problem prevents the running of **shutdown**.

See Also

shutdown(C)

HD(C)

Name

hd - Displays files in hexadecimal format.

Syntax

hd [**-format** [**-s** *offset*] [**-n** *count*] [*file*] . . .

Description

The **hd** command displays the contents of files in hexadecimal, octal, decimal, and character formats. Control over the specification of ranges of characters is also available. The default behavior is with the following flags set: "-abx -A". This says that addresses (file offsets) and bytes are printed in hexadecimal and that characters are also printed. If no *file* argument is given, the standard input is read.

Options include:

-s *offset* Specify the beginning offset in the file where printing is to begin. If no *file* argument is given, or if a seek fails because the input is a pipe, *offset* bytes are read from the input and discarded. Otherwise, a seek error will terminate processing of the current file.

The *offset* may be given in decimal, hexadecimal (preceded by '0x'), or octal (preceded by a '0'). It is optionally followed by one of the following multipliers: **w**, **l**, **b**, or **k**; for words (2 bytes), long words (4 bytes), blocks (512 bytes), or KB (1024 bytes). This is the one case where "b" does not stand for bytes. Because specifying a hexadecimal offset in blocks would result in an ambiguous trailing b, any offset and multiplier may be separated by an asterisk (*).

-n *count* Specify the number of bytes to process. The *count* is in the same format as *offset*, above.

Format Flags

Format flags may specify addresses, characters, bytes, words (2 bytes) or longs (4 bytes) to be printed in hexadecimal, decimal, or octal. Two special formats may also be indicated: text or ASCII. Format and base specifiers may be freely combined and repeated as desired to specify different bases (hexadecimal, decimal or octal) for different output formats (such as addresses and characters). All format flags appearing in a single argument are applied as appropriate to all other flags in that argument.

acbwlA

Output format specifiers for addresses, characters, bytes, words, long words and ASCII respectively. Only one base specifier is be used for addresses; the address appears on the first line of output that begins each new offset in the input.

The character format prints printable characters unchanged, special C escapes as defined in the language, and the remaining values in the specified base.

The ASCII format prints all printable characters unchanged, and all others as a period (.). This format appears to the right of the first of other specified output formats. A base specifier has no meaning with the ASCII format. If no other output format (other than addresses) is given, **bx** is assumed. If no base specifier is given, *all* of **xdo** are used.

xdo Output base specifiers for hexadecimal, decimal and octal. If no format specifier is given, *all* of **acbwl** are used.

t Print a text file, each line preceded by the address in the file. Normally, lines should be terminated by a **\n** character; but long lines are broken up. Control characters in the range 0x00 to 0x1f are printed as '∧@' to '∧_'. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special characters (~ ∧ \) are preceded by a backslash (\) to escape their special meaning. As special cases, two values are represented numerically as \177 and \377. This flag overrides all output format specifiers except addresses.

HEAD(C)

Name

head - Prints the first few lines of a stream.

Syntax

head [-*count*] [*file* ...]

Description

This filter prints the first *count* lines of each of the specified files. If no files are specified, **head** reads from the standard input. If no *count* is specified, then 10 lines are printed.

See Also

tail(C)

Credit

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

ID(C)

Name

id - Prints user and group IDs and names.

Syntax

id

Description

The **id** command writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

See Also

logname(C)

INSTALL(C)

Name

install - Install commands.

Syntax

```
install [-c dira] [-f dirb] [-i] [-n dirc] [-o] [-s] file [ dirx . . . ]
```

Description

The **install** command is most commonly used in "makefiles" to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command file. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* . . .) are given, **install** searches (using **find** (C)) a set of default directories (/bin, /usr/bin, /etc, /lib, and /usr/lib, in that order) for a file with the same name as *file.* When the first occurrence is found, **install** issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* . . .) are specified after *file*, those directories are searched before the directories specified in the default list.

The meanings of the options are:

-c *dira* Installs a new command file in the directory specified in *dira*. Looks for *file* in *dira* and installs it there if it is not found. If it is found, **install** issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **-s** option.

-f *dirb*	Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to 755 and *bin*, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **-o** or **-s** options.
-i	Ignores default directory list, searching only through the given directories (*dirx* . . .). May be used alone or with any other options except **-c** and **-f**.
-n *dirc*	If *file* is not found in any of the searched directories, it put in the directory specified in *dirc*. The mode and owner of the new file will be set to 755 and *bin*, respectively. May be used alone or with any other options except **-c** and **-f**.
-o	If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. May be used alone or with any other options except **-c**.
-s	Suppresses printing of messages other than error messages. May be used alone or with any other options.

See Also

find(C)

JOIN(C)

Name

join - Joins two relations.

Syntax

```
join [options] file1 file2
```

Description

The **join** command forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is a dash (-), the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

-a*n*	In addition to the normal output, produces a line for each unpairable line in file *n*, where *n* is 1 or 2.
-e *s*	Replaces empty output fields by string *s*.
-j*n m*	Joins on the *mth* field of file *n*. If *n* is missing, uses the *m th* field in each file.
-o *list*	Each output line makes up the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number.
-t*c*	Uses character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

See Also

awk(C), comm(C), sort(C)

Comment

With default field separation, the collating sequence is that of **sort -b**; with **-t**, the sequence is that of a plain sort.

KILL(C)

Name

kill - Terminates a process.

Syntax

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

Description

The **kill** conunand sends signal 15 (terminate) to the specified processes. This normally kills processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using **ps**(C).

For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number (*signo*) preceded by - is given as the first argument, that signal is sent instead of the terminate signal. In particular, "kill -9 . . . " is a sure **kill**.

See Also

ps(C), sh(C)

LC(C)

Name

lc - Lists directory contents in columns.

Syntax

```
lc[ -1ACFRabcdfgilmnoqrstux ] name . . .
```

Description

The **lc** command lists the contents of files and directories, in columns. If *name* is a directory name, **lc** lists the contents of the directory; if *name* is a filename, **lc** repeats the filename and any other information requested. Output is given in columns and sorted alphabetically. If no argument is given, the current directory is listed. If several arguments are given, they are sorted alphabetically, but file arguments appear before directories.

Files that are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns.

A stream output format is available in which files are listed across the page, separated by commas. The **-m** option enables this format.

The options are:

-1 Forces an output format with one entry per line.

-A Displays all files, including ".." and those that begin with ".", unless the user is super-user. If the super-user gives this option, ".." and filenames that begin with "." are not displayed.

-C Forces columnar output.

-F Causes 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 an "f".

-R Recursively lists subdirectories.

-a Lists all entries; usually " . " and " .. " are suppressed.

-b Forces printing of nongraphic characters in the \ **ddd** notation, in octal.

-c Sorts by time of file creation.

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

-f Forces each argument to be interpreted as a directory and lists the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.

-g The same as **-l**, except that the owner is not printed.

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

-l Lists in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file. If the file is a special file, instead, the size field contains the major and minor device numbers.

-m Forces stream output format.

-n The same as **-l** switch, but the owner's user ID appears instead of the owner's name. If used with the **-g** switch, the owner's group ID appears instead of the group name.

-o The same as **-l**, except that the group is not printed.

-q Forces printing of nongraphic characters in filenames as the character "?".

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

-s Gives size in 512-byte blocks, including indirect blocks for each entry.

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

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

-x Forces columnar printing to be sorted across rather than down the page.

The mode printed under the **-l** option contains 11 characters. The first character is:

-	If the entry is a plain file.
d	If the entry is a directory.
b	If the entry is a block-type special file.
c	If the entry is a character-type special file.
p	If the entry is a named pipe.
s	If the entry is a semaphore.
m	If the entry is shared data (memory).

The next 9 characters are interpreted as three sets of 3 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 3 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**(C) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

Files

/etc/passwd	To get user IDs for **lc-l**
/etc/group	To get group IDs for **lc-g**

Credit

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

Comments

Newline and tab are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

Column width choices are poor for terminals that can tab.

LINE(C)

Name

line - Copies one line.

Syntax

```
line
```

Description

The **line** command copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on end-of-file and always prints at least a newline. It is often used within shell files to read from the user's terminal.

See Also

sh(C)

LN(C)

Name

ln - Makes a link to a file.

Syntax

```
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 to the file are effective independent of the name by which the file is known.

The **ln** command creates a link to the existing file *name1*. The *name2* argument is a new name referring to the same file contents as *name1*.

It is not allowed to link to a directory or to link across file systems.

See Also

cp(C), mv(C), rm(C)

LOGNAME(C)

Name

logname - Gets login name.

Syntax

```
logname
```

Description

The **logname** command returns the current login name for the user.

Files

/etc/utmp

See Also

env(C), login(M), environ(M)

LOOK(C)

Name

look - Finds lines in a sorted list.

Syntax

```
look [-df] string [file]
```

Description

The **look** command consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options **d** and **f** affect comparisons as in **sort**(C):

-d Dictionary order: only letters, digits, tabs and blanks participate in comparisons.

-f Fold. Uppercase letters compare equal to lowercase.

If no *file* is specified, **/usr/dict/words** is assumed with collating sequence **-df**.

File

usr/dict/words

See Also

sort(C), grep(C)

LPR(C)

Name

lpr - Sends files to the line printer queue for printing.

Syntax

```
lpr [option . . . ] [name . . . ]
```

Description

The **lpr** command 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 filename arguments:

-c	Makes a copy of the file and prints the copy and not the original. Normally files are linked whenever possible.
-r	Removes the file after sending it.
-m	When printing is complete, reports that fact by **mail**(C).
-n	Does not report the completion of printing by **mail**(C). This is the default option.

The file **/etc/default/lpd** contains the setting of the variable BANNERS, which contains the number of pages printed as a banner identifying each printout. This is normally set to either 1 or 2.

Files

/etc/passwd	User's identification and accounting data.
/usr/lib/lpd	Line printer daemon.
/usr/spool/lpd/*	Spool area.
/etc/default/lpd	Contains BANNERS default setting.

See Also

banner(C)

Comment

Once a file has been queued for printing, it should not be changed or deleted until printing is complete. If you want to alter the contents of the file or to remove the file immediately, use the **-c** option to force **lpr** to make its own copy of the file.

LS(C)

Name

ls - Gives information about contents of directories.

Syntax

ls [**-logtasdrucif**] *names*

Description

For each directory named, **ls** lists the contents of that directory; for each file named, **ls** repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents. There are several options:

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

-o The same as **-l** except that the group is not printed.

-g The same as **-l**, except that the owner is not printed.

-t Sorts by time of last modification (latest first) instead of by name.

-a Lists all entries; in the absence of this option, entries whose names begin with a period (.) are not listed.

-s Gives size in 512-byte blocks, including indirect blocks for each entry.

-d If argument is a directory, lists only its name; often used with **-l** to get the status of a directory.

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

-u Uses time of last access instead of last modification for sorting (with the **-t** option) and/or printing (with the **-l** option).

-c Uses time of last modification of the inode (mode, etc.) instead of last modification of the file for sorting (**-t**) and/or printing (**-l**).

-i For each file, prints the inode number in the first column of the report.

-f Forces each argument to be interpreted as a directory and lists the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.

The mode printed under the **-l** option consists of 11 characters. The first character is:

-	If the entry is an ordinary file.
d	If the entry is a directory.
b	If the entry is a block special file.
c	If the entry is a character special file.
p	If the entry is a named pipe.
s	If the entry is a semaphore.
m	If the entry is shared data (memory).

The next 9 characters are interpreted as three sets of 3 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 3 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**(C) for the meaning of this mode. The indications of set-ID and 1000 bit of the mode are capitalized if the corresponding execute permission is not set.

When the sizes of the files in a directory are listed, a total count of blocks including indirect blocks is printed.

Files

/etc/passwd	Gets user IDs for ls -l and ls -o
/etc/group	Gets group IDs for ls -l and ls -g

See Also

chmod(C), find(C), lc(C)

Comment

Newline and tab are considered printing characters in filenames.

MAIL(C)

Name

mail - Sends, reads, or disposes of mail.

Syntax

```
mail [[-u user] [-f mailbox]] [-e] [-R] [-i] [users . . . ]
mail [-s subject] [-i] [users . . . ]
```

Description

The **mail** processing system supports composing of messages, and sending and receiving mail between multiple users. When sending mail, a *user* is the name of a user or of an alias assigned to a machine or to a group of users.

Options include:

-u *user*	Tells **mail** to read the system mailbox belonging to the specified *user.*
-f *mailbox*	Tells **mail** to read the specified *mailbox* instead of the default user's system mailbox.
-e	Allows escapes from compose mode when input comes from a file.
-R	Makes the mail session "read-only" by preventing alteration of the mailbox being read. Useful when accessing system-wide mailboxes.
-i	Tells **mail** to ignore Interrupts (Del) sent from the terminal. This is useful when reading or sending mail over telephone lines where "noise" may produce unwanted Interrupts.

-s *subject* Specifies *subject* as the text of the *subject*: field for the message being sent.

Sending mail

To send a message to one or more other people, invoke **mail** with arguments that are the names of people to send to. You are then expected to type in your message, followed by a Ctrl-D at the beginning of a line.

Reading Mail

To read mail, invoke **mail** with no arguments. This checks your mail out of the system-wide directory so that you can read and dispose of the messages sent to you. A message header is printed out for each message in your mailbox The current message is initially the last numbered message and can be printed using the **print** command (which can be abbreviated **p**). You can move among the messages much as you move between lines in **ed**, with the commands + and - moving backward and forward, and simple numbers typing the addressed message.

If new mail arrives during the mail session you can read in the new messages with the restart command.

Disposing of Mail

After examining a message you can **delete** (**d**) the message or **reply** (**r**) to it. Deletion causes the **mail** program to forget about the message. This is not irreversible, the message can be **undeleted** (**u**) by giving its number, or the **mail** session can be ended by giving the **exit** (**x**) command. If you leave mail with the **quit** (**q**) command, your deleted messages cannot be recovered.

Specifying Messages

Commands such as **print** and **delete** often can be given a list of message numbers as arguments to apply to a number of messages at once. Thus "delete 1 2" deletes messages 1 and 2, while "delete 1-5" deletes messages 1 through 5. The special name * addresses all messages, and $ addresses the last message; thus the command **top**, which prints the first few lines of a message, could be used in **top** * to print the first few lines of all messages.

Replying to or Originating Mail

You can use the **reply** command to set up a response to a message, sending it back to the person who it was from. Text you then type in, up to a Ctrl-D, defines the contents of the message. While you are composing a message, **mail** treats lines beginning with a tilde (~) as special. For instance, typing "~m" (alone on a line) places a copy of the current message into the response, shifting it right by one tabstop. Other escapes set up subject fields, add and delete recipients to the message, and allow you to escape to an editor to revise the message or to a shell to run some commands. (These options are in the summary below.)

Ending a Mail Session

You can end a **mail** session with the **quit (q)** command. Messages can be put in your *mbox* file with the **mbox (mb)** command. If a message is not deleted or mailboxed, it will go back to the post office (/usr/spool/mail/yourname). The **-f** option causes **mail** to read in the contents of your *mbox* (or the specified file) for processing; when you **quit, mail** writes undeleted messages back to this file. The **-i** option causes **mail** to ignore Interrupts (Del).

Using Aliases and Distribution Lists

It is possible to create personal distribution lists so that, for instance, you can send mail to cohorts and have it go to a group of people. Such lists can be defined by placing a line like:

```
  alias cohorts ron bob barry bobo betty beth bobbi
```

in the file *.mailrc* in your home directory. The current list of such aliases can be displayed by the **alias (a)** command in **mail.** System-wide distribution lists can be created by editing /usr/lib/mail/aliases (see **aliases(M)**); these are kept in a slightly different syntax. In mail you send, personal aliases are expanded in mail sent to others so that they will be able to **reply** to the recipients. System wide **aliases** are not expanded when the mail is sent, but any reply returned to the machine will have the system-wide alias expanded.

The **mail** command has a number of options that can be **set** in the *.mailrc* file to alter its behavior; for example, **set askcc** enables the askcc feature. (These options are summarized below.)

Summary

Each **mail** command is typed on a line by itself, and may take arguments following the command word. The command need not be typed in its entirety - the first command that matches the typed prefix is used. For the commands that take message lists as arguments, if no message list is given, the next message forward that satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backward, and if there are no good messages at all, **mail** types "No applicable messages" and ends the command.

-	Goes to the previous message and prints it out. If given a numeric argument *n*, goes to the *n*th previous message and prints it.
+	Goes to the next message and prints it out. If given a numeric argument *n*, goes to the *n*th next message and prints it.
Enter	Goes to the next message and prints it out.
?	Prints a brief summary of commands.
!	Executes the shell command that follows.
=	Prints out the current message number.
+^	Prints out the first message.
$	Prints out the last message.
alias	**(a)** With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, adds the users named in the second and later arguments to the alias named in the first argument.

cd	(**c**) Changes the user's working directory to that specified. If no directory is given, changes to the user's login directory.
delete	(**d**) Takes a list of messages as an argument and marks them all as deleted. Deleted messages are not retained in the system mailbox after a quit, nor are they available to any command other than the **undelete** command.
dp	Deletes the current message and prints the next message. If there is no next message, **mail** says "No more messages".
echo *path*	Expands shell metacharacters.
edit	(**e**) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.
exit	(**x**) Effects an immediate return to the shell without modifying the user's system mailbox, his **mbox** file, or his edit file in **-f**.
file	(**fi**) Prints the name of the file **mail** is reading. If it is a mailbox the name of the owner is returned.
forward	(**f**) Forwards the current message to the named users. Current message is indented within forwarded message.
Forward	(**F**) Forwards the current message to the named users. Current message is not indented within forwarded message.
headers	(**h**) Lists the current range of headers, which is an 18-message group. If a + argument is given, the next 18-message group is printed, and if a - argument is given, the previous 18-message group is printed. Both + and - may take a number to view a particular window. If a message list is given, it prints the specified headers.

hold	(**ho**) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in **mbox.** Use only when the switch **autombox** is set. Does not override the **delete** command.
list	Prints list of **mail** commands.
lpr	(**l**) Prints out each message in a message-list on the line printer.
mail	(**m**) Takes as argument login names and distribution group names and sends mail to those people.
mbox	(**mb**) Marks messages in a message list so that they are saved in the user mailbox after leaving mail.

move *mesg-list mesg-num*

Places the messages specified in *mesg-list* after the message specified in *mesg-num.* If *mesg-num* is 0, *mesg-list* moves to the top of the mailbox.

next	(**n**) Like + or Enter goes to the next message in sequence and prints it. With an argument list, types the next matching message.
print	(**p**) Prints out each message in a message list on the terminal display.
quit	(**q**) Terminates the session, retaining all undeleted, unsaved messages in the system mailbox and removing all other messages. Files marked with a asterisk (*) are saved; files marked with an "M" are saved in the user mailbox. If new mail has arrived during the session, the message "New mail has arrived -- type `restart′ to read." is given. If given while editing a mailbox file with the **-f** flag, the edit file is rewritten. The user returns to the shell, unless the rewrite of edit file fails, in which case the user can escape with the **exit** command.

reply	(r) Takes a message list and sends mail to each message author. The default message must not be deleted.
Reply	(**R**) Takes a message list and sends mail to each message author and each member of the message list in the cc field, just like the **mail** command. The default message must not be deleted.
restart	Reads in messages that arrived during the current mail session.
save	(s) Takes a message list and a filename and appends each message in turn to the end of the file. The filename in quotes, followed by the line count and character count is echoed on the user's terminal.
set	(se) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form *option=value* or *option.*
shell	(sh) Invokes an interactive version of the shell.
size	(si) Takes a message list and prints out the size, in characters, of each message.
source	(so) Reads mail commands from the file given as its only argument.
string *string mesg-list*	Searches for *string* in *mesg-list.* If no *mesg-list* is specified, all undeleted messages are searched. Case is ignored in search.
top	(t) Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to six.
undelete	(u) Takes a message list and marks each one as **not** being deleted.

unset	(**uns**) Takes a list of option names and discards their remembered values; the inverse of **set**.
visual	(**v**) Takes a message list and invokes the visual editor on each message.
write *filename*	(**w**) Saves the body of the message in the named file.

Here is a summary of the compose escapes, which are used when composing messages to perform special functions. Compose escapes are only recognized at the beginning of lines.

+~~ *string*	Inserts the string of text in the message prefaced by a single tilde (~). If you have changed the escape character, then you should double that character instead.
+~?	Prints out help for compose escapes.
+~.	Same as Ctrl-D on a new line.
+~! *cmd*	Executes the indicated shell command, then returns to the message.
+~| *cmd*	Pipes the message through the command as a filter. If the command gives no output or terminates abnormally, retains the original text of the message.
+~_ *mail-command*	Executes a mail command, then returns to compose mode.
+~: *mail-command*	Executes a mail command, then returns to compose mode.
+~**alias**	Prints list of private aliases.
+~**alias** *aliasname*	Prints names included in private aliasname.

+~Alias	Prints list of private, then system-wide aliases for all users named in the current To, CC and Bcc lists.
+~Alias users	Prints list of private, then system-wide aliases for users.
+~b *name . . .*	Adds the given names to the list of blind copy recipients.
+~c *name . . .*	Adds the given names to the list of carbon copy recipients.
+~cc *name . . .*	Same as ~c above.
+~d	Reads the file *dead.letter* from your home directory into the message.
+~e	Invokes the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.
+~h	Edits the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field with the current terminal erase and kill characters.
+~m *mesg-list*	Reads the named messages into the message buffer, shifted right one tab. If no messages are specified, reads the current message.
+~M *mesg-list*	Reads the named messages into the message buffer, shifted right one tab. If no messages are specified, reads the current message.
+~p	Prints out the messages collected so far, prefaced by the message header fields.

+~**q**	Aborts the message being sent, copying the message to *dead.letter* in your home directory if **save** is set.
+~**r** *filename*	Reads the named file into the message buffer.
+~**Return** *name*	Adds the given names to the Return-receipt-to field.
+~**s** *string*	Causes the named string to become the current subject field.
+~**t** *name* . . .	Adds the given names to the direct recipient list.
+~**v**	Invokes a visual editor (defined by the VISUAL option) on the message buffer. After you quit the editor, you may resume appending text to the end of your message.
+~**w** *filename*	Writes the message to the named file.

Options are controlled with the **set** and **unset** commands. An option may be either a switch, in which case it is either on or off, or a string, in which case the actual value is of interest. The switch options include the following:

askcc	Causes you to be prompted for additional copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.
asksubject	Causes **mail** to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field is sent.
autombox	Causes all examined messages to be saved in the user mailbox unless deleted or saved.
autoprint	Causes the **delete** command to behave like **dp**. After deleting a message, the next one will be typed automatically.

chron	Causes messages to be displayed in chronological order.
dot	Permits use of dot (.) as the end of file character when composing messages.
ignore	Causes Interrupt (Del) signals from your terminal to be ignored and echoed as at-signs (@).
mchron	Causes messages to be listed in numerical order (most recently received first), but displayed in chronological order.
metoo	Usually, when a group that contains the sender is expanded, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.
nosave	Prevents aborted messages from being appended to the file **dead.letter** in your home directory on receipt of two Interrupts (Dels) (or a ~ **q**).
quiet	Suppresses the printing of the version header when first invoked.

The following options have string values:

EDITOR	Path name of the text editor to use in the **edit** command and ~e escape. If not defined, a default editor is used.
SHELL	Path name of the shell to use in the ! command and the ~! escape. A default shell is used if this option is not defined.
VISUAL	Path name of the text editor to use in the **visual** command and ~v escape.
escape	If defined, the first character of this option gives the character to use in the place of the tilde (~) to denote escapes.

page=*n*	Specifies the number of lines (*n*) to be printed in a "page" of text when displaying messages.
record	If defined, gives the path name of the file used to record all outgoing mail. If not defined, outgoing mail is not saved.
toplines	If defined, gives the number of lines of a message to be printed out with the **top** command; normally, the first six lines are printed.

Files

/usr/spool/mail/*	System mailboxes.
/usr/name/dead.letter	File where undeliverable mail is deposited.
/usr/name/mbox	Your old mail.
/usr/name/.mailrc	File giving initial mail commands.
/usr/lib/mail/aliases	System-wide aliases.
/usr/lib/mail/aliases.hash	System-wide alias database.
/usr/lib/mail/faliases	Forwarding aliases for the local machine.
/usr/lib/mail/maliases	Machine aliases.
/usr/lib/mail/mailhelp.cmd	Help file.
/usr/lib/mail/mailhelp.esc	Help file.
/usr/lib/mail/mailhelp.set	Help file.
/usr/lib/mail/mailrc	System initialization file.
/usr/bin/mail	The mail command.

See Also

aliases(M), aliashash(M), netutil(C)

Credit

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

MESG(C)

Name

mesg - Permits or denies messages sent to a terminal.

Syntax

```
mesg [n] [y]
```

Description

The **mesg** command with argument **n** prevents messages via **write**(C) by revoking nonuser write permission on the user's terminal. The **mesg** command with argument **y** reinstates permission. By itself, **mesg** reports the current state without changing it.

File

/dev/tty*

See Also

write(C)

Diagnostics

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

MKDIR(C)

Name

mkdir - Makes a directory.

Syntax

mkdir *dirname*

Description

The **mkdir** command creates directories. The standard entries "dot" (.), for the directory itself, and "dot dot" (..), for its parent, are made automatically.

The **mkdir** command requires write permission in the parent directory. The permissions assigned to the new directory are modified by the current file creation mask set by **umask**(C).

See Also

rmdir(C), umask(C)

Diagnostics

The **mkdir** command returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic, and returns nonzero.

MKFS(C)

Name

mkfs - Constructs a file system.

Syntax

```
/etc/mkfs [-y] [-n] special blocks [:inodes] [gap blocks]
/etc/mkfs [-y] [-n] special proto [gap blocks]
```

Description

The **mkfs** command constructs a file system by writing on the special file according to the directions found in the remainder of the command line.

If it appears that the **special** file contains a file system, operator confirmation is requested before overwriting the data. The **-y** "yes" option overrides this, and writes over any existing data without question. The **-n** option causes **mkfs** to terminate without question if the target contains an existing file system. The check used is to read block one from the target device (block one is the super-block) and see whether the bytes are the same. If they are not, this is taken to be meaningful data and confirmation is requested.

If the second argument is given as a string of digits, **mkfs** builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. The boot program is left uninitialized. If the number of inodes is specified, this number should be the same as the estimated number of files in the file system. If the optional number of inodes is not given, the number of inodes is calculated as a function of the system file size.

If the second argument is a file name that can be opened, **mkfs** assumes it to be a prototype file **proto**, and takes its directions from that file. The prototype file contains tokens separated by spaces or newlines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The bootstrap program specified should already be stripped of the header. If the header has not been stripped from the bootstrap program, **mkfs** issues a warning. The second token is a number specifying the size of the created file system. It will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the i-list size in blocks. The next set of tokens is 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**(C)).

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token is a pathname from which the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow that give the major and minor device numbers. If the file is a directory, **mkfs** makes the entries **.** and **..** and (recursively) reads a list of names and file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr  d--777 3 1
     sh   ---755 3 1 /bin/sh
     ken  d--755 6 1
               $
     b0   b--644 3 1 0 0
     c0   c--644 3 1 0 0
     $
$
```

In both command syntaxes, the disk interleaving factors, *gap* and *blocks*, can be specified. The interleaving factors are a function of the disk hardware and are described in detail in the *XENIX Basic Operations Guide*, and *System Administration Manual Appendix A*.

See Also

filesystem(F), dir(F)

Comment

There is no way to specify links when using a prototype file. If the number of inodes is specified on the command line, the maximum number of inodes in the file system is 65500.

MKNOD(C)

Name

mknod - Builds special files.

Syntax

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

/etc/mknod *name* **p**

/etc/mknod *name* **s**

/etc/mknod *name* **m**

Description

The **mknod** command makes a directory entry and corresponding inode for a special file. The first argument is the *name* of the entry. In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (for example, unit, drive, or line number), which may be either decimal or octal.

The assignment of major device numbers is specific to each system.

The **mknod** command can also be used to create named pipes with the **p** option; semaphores with the **s** option; and shared data (memory) with the **m** option.

Only the super-user can use the first form of the syntax.

MKUSER(C)

Name

mkuser - Adds a login ID to the system.

Syntax

/etc/mkuser

Description

The **mkuser** command is used to add more user login IDs to the system. It is the preferred method for adding new users to the system, because it handles all directory creation and password file update. To add a new user to the system, **mkuser** requires four pieces of information: the login name, the initial password, and an optional comment string for the password file. It also allows the new user to be assigned to a group if required, although in most cases a default group is suitable. The program prompts for these four items and validates the given data. The login name is checked against certain criteria (that is, it must be at least three characters and begin with a lowercase letter). The password must follow standard XENIX conventions, see **passwd**(C). The password file comment field can be up to 20 characters of information.

The **mkuser** command takes some of its parameters from a default file, **/etc/default/mkuser**. Currently the two settable options are the path name for the login shell and the root path of home directories. An example default file is:

```
HOME=/usr
```

This file can be edited (by the super-user) to change these defaults. There are three other files in the directory **/usr/lib/mkuser** which may also be altered to suit local options. They are **mkuser.help**, which is the introductory explanation given by **mkuser** on startup, **mkuser.mail**, which is the initial mail message sent to new users, and **mkuser.prof**, the standard **.profile** file given to new users.

The **mkuser** command allocates user IDs starting at 200, or the largest number used in the password file. The default group ID for new users is 50. The minimum group ID allowed for user accounts is 50. The program prompts the operator for an optional group specification. This can either be a numeric group ID, or a group name. If the group exists, the user is added to it. If it does not exist, a new entry in **/etc/group** is created. A new group cannot have a numeric ID less than 51. If a new group is to be created, and the operator only specifies the group name, a free group ID is assigned. Alternatively the operator can specify the group ID too.

The **mkuser** command can only be executed by the super-user.

The minimum length of a legal password, and the minimum and maximum number of weeks used in password aging are specified in **/etc/default/passwd** by the variables PASSLENGTH, MINWEEKS and MAXWEEKS. For example, these variables might be set as follows:

```
PASSLENGTH=6

MINWEEKS=2

MAXWEEKS=6
```

Files

/etc/passwd
/usr/spool/mail/*username*
/etc/default/mkuser
/usr/lib/mkuser/mkuser.help
/usr/lib/mkuser/mkuser.prof
/usr/lib/mkuser/mkuser.mail

See Also

rmuser(C), passwd(C), pwadmin(C)

MORE(C)

Name

more - Views a file one screen full at a time.

Syntax

```
more [-cdflsurw][-n][+linenumber][+/pattern][name ... ]
```

Description

This filter allows examination of continuous text, one screen full at a time. It normally pauses after each screen full, printing "--More--" at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user presses the Space bar, another screen full is displayed. Other possibilities are described below.

The command line options are:

-n An integer that is the size (in lines) of the window which **more** will use instead of the default.

-c The **more** command draws each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while **more** is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

-d The **more** command prompts with the message "Hit space to continue, Rubout to abort" at the end of each screen full. 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 option causes **more** to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended when viewing output that contains unprintable characters, for example, escape sequences. Escape sequences contain characters that would ordinarily occupy screen positions, but that do not print when they are sent to the terminal as part of an escape sequence. Thus, **more** may think that lines are longer than they actually are and fold lines erroneously.

-l Does not treat Ctrl-l (form feed) specially. If this option is not given, **more** pauses after any line that contains a Ctrl-l as if the end of a screen full had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

-s Squeezes multiple blank lines from the output, producing only one blank line. This option maximizes the useful information present on the screen.

-u Normally, **more** handles underlining in a manner appropriate to the particular terminal; if the terminal can perform underlining or has a standout mode, **more** outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The **-u** option suppresses this processing.

-r Normally, **more** ignores control characters that it does not interpret in some way. The **-r** option causes these to be displayed as ∧C where "C" stands for any such character.

-w Normally, **more** exits when it comes to the end of its input. With **-w** however, **more** prompts and waits for any key to be struck before exiting.

+*linenumber*
Starts up at *linenumber*

+/*pattern*
Starts up two lines before the line containing the regular expression *pattern*

The **more** command looks in the file **/etc/termcap** to determine terminal characteristics and to determine the default window size. On a terminal display capable of displaying 24 lines, the default window size is 22 lines.

The **more** command looks in the environment variable **MORE** to preset any flags desired. For example, if you prefer to view files using the **-c** mode of operation, the shell command "MORE=-c" in the **.profile** file causes all invocations of **more** to use this mode.

If **more** is reading from a file, rather than a pipe, a percentage is displayed along with the "--More--" prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences that may be typed when **more** pauses, and their effects, are (*i* is an optional integer argument, defaulting to 1):

i<**space**>	Displays *i* more lines, (or another screen full if no argument is given).
Ctrl-D	Displays 11 more lines (a "scroll"). If *i* is given, the scroll size is set to *i*.
d	Same as Ctrl-D.
*i***z**	Same as typing a space except that *i*, if present, becomes the new window size.
*i***s**	Skips *i* lines and prints a screen full of lines.
*i***f**	Skips *i* screen fulls and prints a screen full of lines.
q or Q	Exits from **more**.
=	Displays the current line number.
v	Starts up the screen editor **vi** at the current line.
h or ?	Help command; gives a description of all the **more** commands.

i/expr	Searches for the *i*th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), the position in the file remains unchanged. Otherwise, a screen full is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.
*i***n**	Searches for the *i*th occurrence of the last regular expression entered.
'	(Single quotation mark) Goes to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.
!*command*	Invokes a shell with *command*. The characters % and ! in "command" are replaced with the current filename and the previous shell command respectively. If there is no current filename, % is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.
*i***:n**	Skips to the *i*th next file given in the command line (skips to last file if *i* doesn't make sense).
*i***:p**	Skips to the *i*th previous file given in the command line. If this command is given in the middle of printing out a file, **more** goes back to the beginning of the file. If *i* doesn't make sense, **more** skips back to the first file. If **more** is not reading from a file, the beep sounds and nothing else happens.
:f	Displays the current filename and line number.
:q or :Q	Exits from **more** (same as **q** or **Q**).
.	Repeats the previous command.

The commands take effect immediately, that is, it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may enter the line kill character to cancel the numerical argument being formed. In addition, the user may enter the erase character to redisplay the "--More--(xx%)" message.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will not show on your terminal display, except for the slash (/) and exclamation (!) commands.

If the standard output is not a teletype, **more** acts just like **cat**, except that a header is printed before each file (if there is more than one).

Files

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

See Also

sh(C), environ(M)

Credit

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

Comment

Before displaying a file, **more** attempts to detect whether it is a nonprintable binary file such as a directory or executable binary image. If **more** concludes that a file is unprintable, it rightly refuses to print it. However, **more** cannot detect all possible kinds of nonprintable files.

MOUNT(C)

Name

mount - Mounts a file structure.

Syntax

/etc/mount [*special-device directory* [**-r**]]

/etc/umount *special-device*

Description

The **mount** command announces to the system that a removable file structure is present on *special-device.* The file structure is mounted on *directory*. The *directory* must already exist; it becomes the name of the root of the newly mounted file structure.

The **mount** and **umount** commands maintain a table of mounted devices. If invoked with no arguments, for each special device **mount** prints the name of the device, the directory name of the mounted file structure, whether the file structure is readonly, and the date it was mounted.

The optional last argument indicates that the file is to be mounted read-only. Physically write-protected files must be mounted in this way or errors occur when access times are updated, whether or not any explicit write is attempted.

The **umount** command removes the removable file structure previously mounted on device *special-device.*

File

/etc/mnttab	Mount table

See Also

umount(C), mnttab(F)

Diagnostics

The **mount** command issues a warning if the file structure to be mounted is currently mounted under another name.

Busy file structures cannot be dismounted with **umount.** A file structure is busy if it contains an open file or some user's working directory.

Comments

Some degree of validation is done on the file structure; however it is generally unwise to mount corrupt file structures.

Be aware that when in single-user mode, the commands that look in **/etc/mnttab** for default arguments (for example **df, ncheck, quot**, **mount**, and **umount**) give either incorrect results (because of a corrupt **/etc/mnttab** from a nonshutdown stoppage) or no results (because of an empty mnttab from a **shutdown** stoppage).

In multiuser mode, this is not a problem; **/etc/rc** initializes **/etc/mnttab** to contain only **/dev/root** and subsequent mounts update it appropriately.

The **mount**(C) and **umount**(C) commands use a lock file to guarantee exclusive access to **/etc/mnttab** , the commands which just read it (those mentioned above) do not, so it is possible to hit a window during which it is corrupt. This is not a problem in practice because **mount** and **umount** are not frequent operations.

MV(C)

mv - Moves or renames files and directories.

Syntax

The **mv** command *file1 file2*

mv *file . . . directory*

mv *directory directory*

Description

The **mv** command moves or changes the name of *file1* to *file2.*

If *file2* already exists, it is erased before *file1* is moved. If *file2* has a mode that prevents writing, **mv** prints the mode 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 filenames.

In the third form, a directory can be renamed only.

The command **mv** refuses to move a file onto itself.

See Also

cp(C), copy(C)

Comment

If *file1* and *file2* lie on different systems, **mv** must copy the file and delete the original. In this case, the owner becomes that of the copying process and any linking relationship with other files is lost.

NCHECK(C)

Name

ncheck - Generates names from inode numbers.

Syntax

ncheck [**-i***numbers*] [**-a**] [**-s**] [*filesystem*]

Description

The **ncheck** command with no argument generates a path name vs. inode number list of all files on the set of file systems specified in **/etc/mnttab**. The two characters / . are appended to the names of directory files. The **-i** option reduces the report to only those files whose inode *numbers* follow. The **-a** option allows printing of the names **.** and **..**, which are ordinarily suppressed. The **-s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy. A single *filesystem* may be specified rather than the default list of mounted file systems.

File

/etc/mnttab

See Also

fsck(C), sort(C)

Diagnostics

When the file system structure is improper, **??** denotes the "parent" of a parentless file and a pathname beginning with **...** denotes a loop.

Comment

See "Comments" under **mount**(C).

NETUTIL(C)

Name

netutil - Administers the XENIX network.

Syntax

```
netutil [-option]
```

Description

The **netutil** command allows the user to create and maintain a network of XENIX systems. A network is a linking over serial lines of two or more XENIX systems. It is used to send mail between systems with the **mail**(C) command, transfer files between systems with the **rcp**(C) command, and execute commands from a remote system with the **remote**(C) command.

The **netutil** command is used to create and distribute the data files needed to implement the network. It is also used to start and stop the network. The *option* argument may be any one of **install**, **save**, **restore**, **start**, **stop**, or the numbers 1 through 5 respectively.

The **install** option interactively creates the data files needed to run the network. The **save** option saves these files on diskettes, allowing them to be distributed to the other systems in the network. The **restore** option copies the data files from diskette back to a system. The **start** option starts the network. The **stop** option stops the network. An *option* may also be any decimal digit in the range 1 to 5. If invoked without an *option*, the command displays a menu from which to choose one. Once an option is selected, it prompts for additional information if needed.

A network must be installed before it can be started. Installation consists of creating appropriate configuration files with the **install** option. This option requires the name of each machine in the network, the serial lines to be used to connect the systems, the speed of transmission for each line, and the names of the users on each system. Once created, the files must be distributed to each computer in the network with the **save** and **restore** options. The network is started by using the **start** option on each system in the network. Once the system starts, mail and remote commands can be passed along the network. A record of the transmissions between computers in a network can be kept in the network log files.

File

/etc/netutil

See Also

aliases(M), aliashash(M), mail(C), micnet(M), remote(C), rcp(C), systemid(M), top(M), *XENIX System Administration*

NEWGRP(C)

Name

newgrp - Logs user in to a new group.

Syntax

```
newgrp [group]
```

Description

The **newgrp** command changes the *group* identification of its caller. The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

The **newgrp** command without an argument changes the *group* identification to the group in the password file; in effect, it changes the *group* identification back to the caller's original *group*.

When most users log in, they are members of the *group* named *group*.

Files

/etc/group

/etc/passwd

See Also

login(M), group(M)

NEWS(C)

Name

news - Print news items.

Syntax

news [-a] [-n] [-s] [items]

Description

The **news** command is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, **news** prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. The **news** command stores the "currency" time as the modification date of a file named **.news__time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

The **-a** option causes **news** to print all items, regardless of currency. In this case, the stored time is not changed.

The **-n** option causes **news** to report the names of the current items without printing their contents, and without changing the stored time.

The **-s** option causes **news** to report how many current items exist, without printing their names or contents and without changing the stored time.

All other arguments are assumed to be specific news items that are to be printed.

If the Interrupt (Del) key is struck during the printing of a news item, printing stops and the next item is started. Another Interrupt (Del) within one second of the first causes the program to terminate.

Files

/usr/news/*
$HOME/.news_time

See Also

profile(M), environ(M)

NICE(C)

Name

nice - Runs a command at a different priority.

Syntax

nice [-*increment*] *command* [*arguments*]

Description

The **nice** command executes *command* with a lower CPU scheduling priority. Priorities range from 0 to 39, where 0 is the highest priority and 39 is the lowest. By default, commands have a priority of 20. If an *-increment* argument is given where *increment* is in the range 1-19, *increment* is added to the default priority of 20 to produce a numerically higher priority, meaning a **lower** scheduling priority. If no *increment* is given, an increment of 10 to produce a priority of 30 is assumed.

The super-user may run commands with priority **higher** than normal by using a double negative increment. For example, an argument of **--10** would decrement the default to produce a priority of 10, which is a higher scheduling priority than the default of 20.

See Also

nohup(C)

Diagnostic

The **nice** command returns the exit status of the subject command.

Comment

An *increment* larger than 19 is equivalent to 19.

NL(C)

Name

nl - Adds line numbers to a file.

Syntax

nl [**-b***type*] [**-h***type*] [**-f***type*] [**-v***start#*] [**-i***incr*] [**-p**] [**-l***num*] [**-s***sep*] [**-w***width*] [**-n***format*] *file*

Description

The **nl** command reads lines from the named *file*, or the standard input if no *file* is named, and reproduces the lines on the standard output. Lines are numbered on the left, according to the command options in effect.

The **nl** command views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (for example, no numbering of header and footer lines, while numbering blank lines only in the body).

The start of logical page sections is signaled by input lines containing nothing but the following characters:

Page Section	*Line Contents*
Header	\:\:\:
Body	\:\:
Footer	\:

Unless signaled otherwise, **nl** assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional filename. Only one file may be named. The options are:

-b*type*	Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p***string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).
-h*type*	Same as **-b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).
-f*type*	Same as **-b***type* except for footer. Default *type* for logical page footer is **n** (no lines numbered).
-v*start#*	The *start#* is the initial value used to number logical page lines. Default is **1**.
-i*incr*	The *incr* is the increment value used to number logical page lines. Default is **1**.
-p	Does not restart numbering at logical page delimiters.
-l*num*	The *num* is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ba**, **-ba**, and/or **-fa** option is set). Default is **1**.

-s*sep*	The *sep* is the character used in separating the line number and the corresponding text line. Default *sep* is a tab.
-w*width*	The *width* is the number of characters to be used for the line number. Default *width* is **6**.
-n*format*	The *format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

See Also

pr(C)

NOHUP(C)

Name

nohup - Runs a command immune to hangups and quits.

Syntax

nohup *command* [*arguments*]

Description

The **nohup** command executes *command* with hangups and quits ignored. If output is not redirected by the user, it will be sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

See Also

nice(C)

OD(C)

Name

od - Displays files in octal format.

Syntax

od [-bcdox] [*file*] [[+]*offset*[.] [**b**]]

Description

The **od** command displays *file* in one or more formats as selected by the first argument. If the first argument is missing, **-o** is default. The meanings of the format options are:

-b	Interprets bytes in octal.
-c	Interprets bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**\Ø,** backspace=**\b** , form feed=**\f** , new line=**\n** , return=**\r** , tab=**\t** ; others appear as three-digit octal numbers.
-d	Interprets words in decimal.
-o	Interprets words in octal.
-x	Interprets words in hexadecimal.

The *file* argument specifies which file is to be displayed. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where displaying is to start. This argument is normally interpreted as octal bytes. If **.** is appended, the *offset* is interpreted in decimal. If **b** is appended, the *offset* is interpreted in blocks of 512 bytes. If the file argument is omitted, the *offset* argument must be preceded by +.

The display continues until end-of-file.

See Also

hd(C)

PACK(C)

Name

pack, pcat, unpack - Compresses and expands files.

Syntax

pack [-] *name* . . .

pcat *name* . . .

unpack *name* . . .

Description

The **pack** command attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.**z** with the same access modes, access and modified dates, and owner as those of *name*. If **pack** is successful, *name* is removed. Packed files can be restored to their original form using **unpack** or **pcat**.

The **pack** command uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of *name* cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or graphics.

Text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

The **pack** command returns a value that is the number of files that it failed to compress.

No packing occurs if:

- The file appears to be already packed.
- The filename has more than 12 characters.
- The file has links.
- The file is a directory.
- The file cannot be opened.
- No disk storage blocks are saved by packing.
- A file called *name*.**z** already exists
- The **.z** file cannot be created.
- An I/O error occurred during processing.

The last segment of the filename must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

The **pcat** command does for packed files what **cat**(C) does for ordinary files. The specified files are unpacked and written to the standard output. To view a packed file named *name*.**z** use:

pcat *name*.**z**

or just:

pcat *name*

To make an unpacked copy, with the name *nnn*, of a packed file named **name.z** (without destroying **name.z**) use the command:

pcat *name* >*nnn*

The **pcat** command returns the number of files it was unable to unpack. Failure may occur if:

- The filename (exclusive of the **.z**) has more than 12 characters.

- The file cannot be opened.

- The file does not appear to be the output of **pack**.

The **unpack** command expands files created by **pack**. For each file *name* specified in the command, a search is made for a file called *name*.**z** (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

The **unpack** command returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in **pcat**, as well as in a file where the unpacked name already exists, or if the unpacked file cannot be created.

PASSWD(C)

Name

passwd - Changes login password.

Syntax

```
passwd name
```

Description

This command changes (or installs) a password associated with the login *name*.

The program prompts for the old password (if any) and then for the new one (twice). The user must supply these. Passwords can be of any reasonable length, but only the first eight characters of the password are significant. The minimum number of characters allowed in a new password is determined by the PASSLENGTH variable. Although the minimum can be three, a minimum of five is strongly recommended, because passwords shorter that five are much easier to guess or discover by trial and error.

Only the owner of the *name* or the super-user may change a password; the owner must prove he knows the old password. Only the super-user can create a null password.

The password file is not changed if the new password is the same as the old password, or if the password has not "aged" sufficiently; (see **passwd**(M)).

The minimum length of a legal password, and the minimum and maximum number of weeks used in password aging are specified in **/etc/default/passwd** by the variables PASSLENGTH, MINWEEKS and MAXWEEKS. For example, these variables might be set as follows:

```
PASSLENGTH=6
MINWEEKS=2
MAXWEEKS=6
```

MINWEEKS and MAXWEEKS values must be in the range 0 to 63. If PASSLENGTH is not in the range 3 to 8, it is set to 5.

Files

/etc/default/passwd
/etc/passwd

See Also

login(M), pwadmin(C), default(M), passwd(M)

PR(C)

Name

pr - Prints files on the standard output.

Syntax

```
pr [options] [files]
```

Description

The **pr** command prints the named files on the standard output. If *file* is -, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines that do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until **pr** has completed printing.

Options may appear singly or be combined in any order. Their meanings are:

+*k*	Begins printing with page *k* (default is 1).
-*k*	Produces *k*-column output (default is 1). The options **-e** and **-i** are assumed for multi-column output.
-a	Prints multi-column output across the page.
-m	Merges and prints all files simultaneously, one per column (overrides the **-***k*, and **-a** options).
-d	Double-spaces the output.

-e*ck*	Expands input tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any nondigit character) is given, it is treated as the input tab character (default for *c* is the tab character).
-i*ck*	In output, replaces whitespace wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any nondigit character) is given, it is treated as the output tab character (default for *c* is the tab character).
-n*ck*	Provides *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **-m** output. If *c* (any nondigit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
-w*k*	Sets the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).
-o*k*	Offsets each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
-l*k*	Sets the length of a page to *k* lines (default is 66).
-h	Uses the next argument as the header to be printed instead of the filename.
-p	Pauses before beginning each page if the output is directed to a terminal (**pr** beeps and waits for a carriage return).
-f	Uses form feed character for new pages (default is to use a sequence of linefeeds). Pauses before beginning the first page if the standard output is associated with a terminal.

-r Prints no diagnostic reports on failure to open files.

-t Prints neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quits printing after the last line of each file without spacing to the end of the page.

-s*c* Separates columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

Examples

The following prints **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

The following writes **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr -e9 -t <file1>file2
```

See Also

cat(C)

PS(C)

Name

ps - Reports process status.

Syntax

ps [*options*]

Description

The **ps** command prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. Otherwise, the information that is displayed is controlled by the following *options*:

-e Prints information about all processes.

-d Prints information about all processes, except process group leaders.

-a Prints information about all processes, except process group leaders and processes not associated with a terminal.

-f Generates a *full* listing. (Normally, a short listing containing only process ID, terminal (tty) identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.

-l Generates a *long* listing. See below.

-c *corefile* Uses the file *corefile* in place of **/dev/mem**.

-s *swapdev* Uses the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*.

-n *namelist* The argument is taken as the name of an alternate *namelist* (**/xenix** is the default.)

-t *tlist*	Restricts listing to data about the processes associated with the terminals given in *tlist*, where *tlist* can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.
-p *plist*	Restricts listing to data about processes whose process ID numbers are given in *plist*, where *plist* is in the same format as *tlist*.
-u *ulist*	Restricts listing to data about processes whose user ID numbers or login names are given in *ulist*, where *ulist* is in the same format as *tlist*. In the listing, the numerical user ID is printed unless the **-f** option is used, in which case the login name is printed.
-g *glist*	Restricts listing to data about processes whose process groups are given in *glist*, where *glist* is a list of process group leaders and is in the same format as *tlist*.

The column headings and the meaning of the columns in a **ps** listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; **all** means that the heading always appears. These two options only determine what information is provided for a process; they do not determine which processes will be listed.

F (l) A status word consisting of flags associated with the process. Each flag is associated with a bit in the status word. These flags are added to form a single octal number. Process flag bits and their meanings are:

Ø1	in core;
Ø2	system process;
Ø4	locked in core (for example, for physical I/O);
1Ø	being swapped;
2Ø	being traced by another process.

S	**(l)**	The state of the process:
		
		∅ non-existent;
		S sleeping;
		W waiting;
		R running;
		I intermediate;
		Z terminated;
		T stopped.

UID	**(f,l)**	The user ID number of the process owner; the login name is printed under the **-f** option.
PID	**(all)**	The process ID of the process; it is possible to end a process if you know this data.
PPID	**(f,l)**	The process ID of the parent process.
C	**(f,l)**	Processor utilization for scheduling.
STIME	**(f)**	Starting time of the process.
PRI	**(l)**	The priority of the process; higher numbers mean lower priority.
NI	**(l)**	Nice value; used in priority computation.
ADDR	**(l)**	The memory address of the process, if resident; otherwise, the disk address.
SZ	**(l)**	The size in blocks of the core image of the process, but not including the size of text shared with other processes. Because this size includes the current size of the stack, it will vary as the stack size varies.
WCHAN	**(l)**	The event for which the process is waiting or sleeping; if blank, the process is running.
TTY	**(all)**	The controlling terminal for the process.
TIME	**(all)**	The cumulative execution time for the process.

CMD	**(all)**	The command name; the full command name and its arguments are printed under the **-f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **-f** option, **ps** tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **-f** option, is printed in square brackets.

Files

/xenix	system namelist
/dev/mem	memory
/dev	searched to find swap device and terminal (tty) names.

See Also

kill(C), nice(C)

Comments

System conditions can change while **ps** is running; the picture it gives is only a close approximation to reality.

Some data printed for defunct processes are irrelevant.

PSTAT(C)

Name

pstat - Reports system information.

Syntax

```
pstat [-aixpf] [-u ubase] [-c corefile] [file]
```

Description

The **pstat** command 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 **/xenix**. Options are:

-a In conjunction with the **-p** option, describes all process slots rather than just active ones.

-i Prints 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 **filesystem**(F) must be corrected
	A Access time must be corrected
	M File system is mounted here
	W Wanted by another process (L flag is on)
	T Contains a text file
	C Changed time must be corrected
CNT	Number of open file table entries for this mode.
DEV	Major and minor device number of file system in which this inode resides.
INO	I-number within the device.
MODE	Mode bits.
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 Prints the text table with these headings:

LOC	The core location of this table entry.
FLAGS	Miscellaneous state variables encoded thus:
	T **ptrace** 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 BSIZE bytes.
CADDR	Core address, measured in units of memory management resolution.
SIZE	Size of text segment, measured in units of memory management resolution.
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 Prints process table for active processes with these headings:

LOC	The core location of this table entry.
S	Run state encoded as follows:
	Ø No process
	1 Waiting for some event
	3 Runnable
	4 Being created
	5 Being terminated
	6 Stopped under trace
F	Miscellaneous state variables, ORed together:
	01 Loaded
	02 The scheduler process
	04 Locked
	010 Swapped out
	020 Traced
	040 Used in tracing
	0100 Locked in
PRI	Scheduling priority.

SIGNAL	Signals received (signals 1-16 coded in bits 0-15)
UID	Real user ID.
TIM	Time resident in seconds; times > 127 coded as 127.
CPU	Weighted integral of CPU time, for scheduler.
NI	Nice level.
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 units of memory management resolution. If swapped out, the position in the swap area measured in multiples of BSIZE bytes.
SIZE	Size of process image, measured in units of memory management resolution.
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** measured in seconds.

-f Print the open file table with these headings:

LOC	The core location of this table entry.
FLG	Miscellaneous state variables encoded as follows: **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.

-u *ubase*

Print information about a user process; the *hexaddr* argument is its hexadecimal address. The address can be displayed using the **ps**(C) command. The user process must be in main memory, or the file used can be a core image and the address **0**.

-c *corefile*

Use the file *corefile* in place of **/dev/mem**.

-n *namelist*

Use the file *namelist* as an alternate namelist in place of **/xenix**.

Files

/xenix	Namelist
/dev/mem	Default source of tables

See Also

ps(C), filesystem(F)

PWADMIN(C)

Name

pwadmin - Performs password aging administration.

Syntax

pwadmin -dcfan [**-min** *weeks*] [**-max** *weeks*] *user*

Description

The **pwadmin** command is used to examine and modify the password aging information in the password file. The options one can specify are:

-d Displays the password aging information.

-f Forces the user to change his password at the next login.

-c Prevents the user from changing his password.

-a Enables password aging for the given user. This option sets the minimum number of weeks that the user must wait before changing his password and the maximum number of weeks that a user can keep his current password to the values defined by the MINWEEKS and MAXWEEKS variables in the **/etc/default/passwd** file. If the file is not found or the defined values are not in range 0 to 63, the default values 2 and 4 are used.

-n Disables the password aging feature.

-min Uses the next argument as the minimum number of weeks before the user can change his password. (This prevents him from changing his password back to the old one).

-max Uses the next argument as the number of weeks before the user must change his password again.

File

/etc/passwd

See Also

passwd(C), passwd(**M**)

Comments

The user must not attempt to force a new password by setting both the **-min** and **-max** values to zero. To force a password, use the **-f** option.

The user must not attempt to prevent further password changes by setting the **-min** value greater than the **-max** value. To prevent changes, use the **-c** option.

PWCHECK(C)

Name

pwcheck - Checks password file.

Syntax

pwcheck [*file*]

Description

The **pwcheck** command scans the password file and checks for any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is **/etc/passwd**.

File

/etc/passwd

See Also

grpcheck(C), group(M), passwd(M)

PWD(C)

Name

pwd - Prints working directory name.

Syntax

pwd

Description

The **pwd** command prints the path name of the working (current) directory.

See Also

cd(C)

Diagnostics

"Cannot open .." and "Read error in .." indicate possible file system trouble. In such cases, see the *XENIX System Administration* for information on fixing the file system.

QUOT(C)

Name

quot - Summarizes file system ownership.

Syntax

quot [*option*] . . . [*filesystem*]

Description

The **quot** command prints the number of blocks in the named *filesystem* currently owned by each user. If no *filesystem* is named, the file systems given in **/etc/mnttab** are examined.

The following options are available:

-n Causes the following pipeline to produce a list of all files and their owners:

```
ncheck filesystem | sort +0n | quot -n filesystem
```

-c Prints three columns, giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file. Data for files of size greater than 499 blocks are included in the figures for files of exactly size 499.

-f Prints count of number of files as well as space owned by each user.

Files

/etc/passwd	Gets user names
/etc/mnttab	Contains list of mounted file systems

See Also

du(C), ls(C)

Comments

Holes (empty spaces) in files are counted as if they actually occupied space.

See also "Comment" under **mount**(C).

RANDOM(C)

Name

random - Generates a random number.

Syntax

random **[-s]** [*scale*]

Description

The **random** command generates a random number on the standard output and returns the number as its exit value. By default, this number is either 0 or 1; that is, *scale* is 1 by default. If *scale* is given a value between 1 and 255, the range of the random value is from 0 to *scale.* If *scale* is greater than 255 an error message is printed.

When the **-s** (silent) option is given, the random number is returned as an exit value but is not printed on the standard output. If an error occurs, **random** returns an exit value of zero.

Comments

This command does not perform any floating point computations.

The **random** command uses the time of day as a seed.

RCP(C)

Name

rcp - Copies files across XENIX systems.

Syntax

rcp [*options*] [*srcmachine*:]*srcfile* [*destmachine*:]*destfile*

Description

The **rcp** command copies files between systems in a Micnet network. The command copies the *srcmachine*:*srcfile* to *destmachine*:*destfile*, where *srcmachine*: and *destmachine*: are optional names of systems in the network, and *srcfile* and *destfile* are path names of files. If a machine name is not given, the name of the current system is assumed. If **-** is given in place of *srcfile*, **rcp** uses the standard input as the source. Directories named on the destination machine must be publicly writable. Directories and files on a remote source machine must be publicly readable.

The available options are:

-m

Mails and reports completion of the command, whether there is an error or not.

-u[*machine*:]*user*

Any mail goes to the named *user* on *machine*. The default *machine* is the system on which **rcp** is invoked.

The **rcp** command is useful for transferring small numbers of files across the network. The network consists of daemons that periodically awaken and send files from one system to another. The network must be installed using **netutil**(C) before **rcp** can be used. Also, to enable transfer of files from a remote system, the line:

```
rcp=/usr/bin/rcp
```

or

```
executeall
```

must be added to the default file **/etc/default/micnet** on the systems in the network.

Example

```
rcp -m machine1:/etc/mnttab /tmp/vtape
```

See Also

netutil(C), remote(C), mail(C), micnet(**M**)

Diagnostics

If an error occurs, mail is sent to the user.

Comments

Full path names must be specified for remote files.

The **rcp** command handles binary data files transparently, no extra switches or protocols are needed to handle them. Wild cards are not expanded on the remote machine.

REMOTE(C)

Name

remote - Executes commands on a remote XENIX system.

Syntax

remote [**-**][**-f** *file*][**-m**][**-u** *user*] *machine command* [*arguments*]

Description

The **remote** command is a limited networking facility that permits execution of XENIX commands across serial lines. Commands on any connected system may be executed from the host system using **remote**. A command line consisting of *command* and any blank-separated *arguments* is executed on the remote *machine*. A machine's name is located in the file **/etc/systemid**. Wild cards are not expanded on the remote machine, so they should not be specified in *arguments*. The optional **-m** switch causes mail to be sent to the user telling whether the command is successful.

The available options follow:

-	A dash signifies that standard input is used as the standard input for *command* on the remote *machine*. Standard input comes from the local host and not from the remote machine.
-f *file*	Use the specified *file* as the standard input for *command* on the remote *machine*. The *file* exists on the local host and not on the remote system.
-m	Mails the user a report of command completion. By default, mail reports only errors.

-u *user* Any report mail goes to the named user rather than to the executor of the command. The user name may have a *machine* name to signify a user on some remote system.

Before **remote** can be successfully used, a network of systems must first be set up and the proper daemons initialized using **netutil**(C). Also, entries for the command to be executed using **remote** must be added to the **/etc/default/micnet** files on each remote machine.

Example

The following command executes an **ls** command on the remote directory **/tmp** of the machine *machine1*:

```
remote -m machine1 ls /tmp
```

See Also

rcp(C), mail(C), netutil(C), micnet(M)

Comment

The **mail** command uses the equivalent of **remote** to send mail between systems.

RESTORE(C)

Name

restore - Invokes incremental file system restorer.

Syntax

```
restore key [arguments]
```

Description

The **restore** command reads archive media backed up with the **backup**(C) command. The *key* specifies what is to be done. *Key* is one of the characters **rRxXtT**, optionally combined with **f**.

f Uses the first *argument* as the name of the archive instead of the default.

r,R The archive is read and loaded into the file system specified in *argument*. If the key is **R**, **restore** asks which archive of a multivolume set to start on. This allows **restore** to be interrupted and then restarted (an **fsck** must be done before the restart).

x,X Each file on the archive named by an *argument* is extracted. The filename has all "mount" prefixes removed; for example, if **/usr** is a mounted file system, **/usr/bin/lpr** is named **/bin/lpr** on the archive. The extracted file is placed in a file with a numeric name supplied by **restore** (actually the inode number). To keep the amount of archive read to a minimum, the following procedure is recommended:

1. Mount volume 1 of the set of backup archives.

2. Type the **restore** command.

3. The **restore** command announces whether it found the files, gives the numeric name that it assigns to the file, and in the case of a tape, rewinds to the start of the archive.

4. It then asks you to "mount the desired tape volume." Type the number of the volume you choose. On a multivolume backup, the recommended procedure is to mount the last through the first volumes, in that order. The **restore** command checks to see if any of the requested files are on the mounted archive (or a later archive, thus thc reverse order). If the requested files are not there, **restore** doesn't read through the tape. If you are working with a single-volume backup or if the number of files being restored is large, respond to the query with **1** and **restore** will read the archives in sequential order.

t Prints the date the archive was written and the date the file system was backed up.

T Same as **t**, in addition, T returna a listing of the files names contained in the backup.

The **r** option should only be used to restore a complete backup archive onto a clear file system or to restore an incremental backup archive onto a file system so created. The following:

```
/etc/mkfs /dev/hdØ3 10000
restor r /dev/hdØ3
```

is a typical sequence to restore a complete backup. Another **restore** can be done to get an incremental backup in addition to this.

A **backup** followed by a **mkfs** and a **restore** is used to change the size of a file system.

Files

rst*	Temporary files
/etc/default/dump	Name of default archive device

The default archive unit varies with installation.

See Also

backup(C), fsck(C), mkfs(C)

Diagnostics

Various diagnostics are involved with reading the archive and writing the diskette. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the backup.

If the backup extends over more than one diskette or tape, it may ask you to change diskettes or tapes. Press Enter when the next unit has been mounted.

Comment

It is not possible to successfully **restore** an entire active root file system.

RM(C)

Name

rm, rmdir - Removes files or directories.

Syntax

rm [**-fri**] *file* . . .

rmdir *dir* . . .

Description

The **rm** command removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with y, the file is deleted, otherwise the file remains. No questions are asked when the **-f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used. In that case, **rm** recursively deletes the entire contents of the specified directory, and the directory itself.

If the **-i** (interactive) option is in effect, **rm** asks whether to delete each file, and if the **-r** option is in effect, whether to examine each directory.

The **rmdir** command removes empty directories.

See Also

rmdir(C)

Diagnostics

Generally self-explanatory. It is prohibited to remove the file **..** to avoid the consequences of inadvertently doing something like:

```
rm -r .*
```

It is also prohibited to remove the root directory of a given file system.

No more than 17 levels of subdirectories can be removed using the **-r** option.

RMDIR(C)

Name

rmdir - Removes directories.

Syntax

rmdir *dir* . . .

Description

The **rmdir** command removes the entries for one or more subdirectories from a directory. A directory must be empty before it can be removed. The **rmdir** command enforces a standard and safe procedure for removing a directory. The **rm -r** *dir* command is a more dangerous alternative to **rmdir.**

The **rmdir** command removes entries for the named directories, which must be empty.

See Also

rm(C)

Comment

The **rmdir** command refuses to remove the root directory of a mounted file system.

RMUSER(C)

Name

rmuser - Removes a user from the system.

Syntax

```
/etc/rmuser
```

Description

The **rmuser** program removes users from the system. It begins by prompting for a user name; after receiving a valid user name as a response, it deletes the named user's entry in the password file, and removes the user's mailbox file, the **.profile** file, and the entire home directory. It also removes the users group entry in **/etc/group** if the user was the only remaining member of that group, and the group ID was greater than 50.

Before removing a user ID from the system, make sure its mailbox is empty and that all files belonging to that user ID have been saved or deleted as required.

The **rmuser** program refuses to remove a user ID or any of its files if one or more of the following checks fails:

- The user name given is one of the "system" user names such as root, sys, sysinfo, cron, or uucp. All user IDs less than 200 are considered reserved for system use and cannot be removed using **rmuser**. Likewise all group IDs less than 50 are not removable using **rmuser**.

- The user's mailbox exists and is not empty.

- The user's home directory contains files other than **.profile**.

The **rmuser** program can only be executed by the super-user.

Files

/etc/passwd
/usr/spool/mail/*username*
$HOME

See Also

mkuser(C), backup(C)

RSH(C)

Name

rsh - Invokes a restricted shell (command interpreter).

Syntax

```
rsh [flags] [name [arg1 . . . ] ]
```

Description

The **rsh** command is a restricted version of the standard command interpreter **sh**(C). It is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of **rsh** are identical to those of **sh**, except that changing directory with **cd**, setting the value of $PATH, using command names containing slashes, and redirecting output using > and >> are all disallowed.

When invoked with the name **-rsh**, rsh reads the user's **.profile** (from **$HOME/.profile**). It acts as the standard **sh** while doing this, except that an interrupt causes an immediate exit, instead of causing a return to command level. The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, **rsh** invokes **sh** to execute it. Thus, it is possible to provide to the end user shell procedures that have access to the full power of the standard shell, while restricting him to a limited menu of commands; this scheme assumes that the end user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions, then leaving the user in an appropriate directory (probably not the login directory).

The **rsh** command is actually just a link to **sh** and any *flags* arguments are the same as for **sh**(C).

The system administrator often sets up a directory of commands that can be safely invoked by **rsh**.

See Also

sh(C), profile(M)

SDDATE(C)

Name

sddate - Prints and sets backup dates.

Syntax

sddate [*name lev date*]

Description

If no argument is given, the contents of the backup date file **/etc/ddate** are printed. The backup date file is maintained by **backup**(C) and contains the date of the most recent backup for each backup level for each file system.

If arguments are given, an entry is replaced or made in **/etc/ddate**. *name* is the last component of the device pathname; *lev* is the backup level number (from 0 to 9); and *date* is a time in the form taken by **date**(C):

```
mmddhhmm[yy]
```

Where the first *mm* is a two-digit month in the range 01-12, *dd* is a two-digit day of the month, *hh* is a two-digit military hour from 00-23, and the final *mm* is a two-digit minute from 00-59. An optional two-digit year, *yy*, is presumed to be an offset from the year 1900, that is, 19*yy*

Some sites may wish to back up file systems by copying them verbatim to backup media. The **sddate** command could be used to make a level 0 entry in **/etc/ddate**, which would then allow incremental backups.

For example:

```
sddate rhd03 5 10081520
```

makes an **/etc/ddate** entry showing a level 5 backup of /dev/rhd03 on October 8, at 3:20 p.m.

File

/etc/ddate

See Also

backup(C), date(C)

Diagnostics

If the date set is syntactically incorrect: bad conversion.

SDIFF(C)

Name

sdiff - Compares files side-by-side.

Syntax

sdiff [*options* ...] *file1 file2*

Description

The **sdiff** command uses the output of **diff**(C) to produce a side-by-side listing of two files indicating the lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are partially different.

For example:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

The following options exist:

-w *n* Uses the next argument, *n*, as the width of the output line. The default line length is 130 characters.

-l Only prints the left side of any lines that are identical.

-s Does not print identical lines.

-o *output*	Uses the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by **diff**(C), are printed if a set of differences shares a common gutter character. After printing each set of differences, **sdiff** prompts the user with a **%** and waits for one of the following user-typed commands:

l	Appends the left column to the output file
r	Appends the right column to the output file
s	Turns on silent mode; does not print identical lines
v	Turns off silent mode
e l	Calls the editor with the left column
e r	Calls the editor with the right column
e b	Calls the editor with the concatenation of left and right
e	Calls the editor with a zero length file
q	Exits from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

See Also

diff(C), ed(C)

SED(C)

Name

sed-Invokes the stream editor

Syntax

sed [**-n**] [**-e** *script*] [**-f** *sfile*] [*files*]

Description

The **sed** command copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **-f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **-e** option and no **-f** options, the flag **-e** may be omitted. The **-n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[*address*[, *address*]] *function* [*arguments*]

In normal operation, **sed** cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, that is, a */regular expression/* in the style of **ed**(C) modified as follows:

- In a context address, the construction \ *?regular expression?*, where *?* is any character, is identical to */regular expression/* Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

- The escape sequence **\n** matches a newline embedded in the pattern space.

- A period (.) matches any character except the terminal newline of the pattern space.

- A command line with no addresses selects every pattern space.

- A command line with one address selects each pattern space that matches the address.

- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to nonselected pattern spaces by use of the negation function **!** (see below).

In the following list of functions, the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with backslashes to hide the newlines. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a+\ *text*

Appends *text*, placing it on the output before reading the next input line.

(2) b *label*
Branches to the : command bearing the *label*. If *label* is empty, branches to the end of the script.

(2) c+\ *text*
Changes text by deleting the pattern space and then appending *text*. With zero or one address or at the end of a two-address range, places *text* on the output and starts the next cycle.

(2) d Deletes the pattern space and starts the next cycle.

(2) D Deletes the initial segment of the pattern space through the first newline and starts the next cycle.

(2) g Replaces the contents of the pattern space with the contents of the hold space.

(2) G Appends the contents of the hold space to the pattern space.

(2) h Replaces the contents of the hold space with the contents of the pattern space.

(2) H Appends the contents of the pattern space to the hold space.

(1) i+\ *text*
Insert. Places *text* on the standard output.

(2) l Lists the pattern space on the standard output with non-printing characters spelled in two-digit ASCII and long lines folded.

(2) n Copies the pattern space to the standard output. Replaces the pattern space with the next line of input.

(2) N Appends the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2)p Prints (copies) the pattern space on the standard output.

(2)P Prints (copies) the initial segment of the pattern space through the first newline to the standard output.

(1)q Quits **sed** by branching to the end of the script. No new cycle is started.

(2)r *rfile*
Reads the contents of *rfile* and places them on the output before reading the next input line.

(2)s/*regular expression*/*replacement*/*flags*
Substitutes the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of / . For a more detailed description see **ed**(C). *Flags* is zero or more of:

g Globally substitutes for all non-overlapping instances of the *regular expression* rather than just the first one.

p Prints the pattern space if a replacement was made.

w *wfile*
Writes the pattern space to *wfile* if a replacement was made.

(2) t *label*
Branches to the colon (:) command bearing *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t** command. If *label* is empty, **t** branches to the end of the script.

(2) w *wfile*
Writes the pattern space to *wfile.*

(2) x Exchanges the contents of the pattern and hold spaces.

(2) y/*string1* /*string2*/

Replaces all occurrences of characters in *string1* with the corresponding characters in *string2*. The lengths of *string1* and *string2* must be equal.

(2) ! *function*

Applies the *function* (or group, if *function* is {) only to lines *not* selected by the addresses.

(0) : *label*

This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) = Places the current line number on the standard output as a line.

(2) { Executes the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

See Also

awk(C), ed(C), grep(C)

SETMNT(C)

Name

setmnt - Establishes /etc/mnttab table.

Syntax

/etc/setmnt

Description

The **setmnt** command creates the **/etc/mnttab** table (see **mnttab**(F)), which is needed for both the **mount**(C) and **umount**(C) commands. The **setmnt** command reads the standard input and creates a **mnttab** entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's *special file* (for example, "hd0.") and *node* is the root name of that file system. Thus, *filesys* and *node* become the first two strings in the **mnttab**(F) entry.

File

/etc/mnttab

See Also

mnttab(F)

Comments

If *filesys* or *node* are longer than 128 characters, errors can occur.

The **setmnt** command enforces an upper limit on the maximum number of **mnttab** entries.

The **setmnt** command is normally invoked by **/etc/rc** when the system starts up.

SETTIME(C)

Name

settime - Changes the access and modification dates of files.

Syntax

settime *mmddhhmm* [*yy*] [**-f** *fname*]*name* . . .

Description

The **settime** command sets the access and modification dates for one or more files. The dates are set to the specified date or to the access and modification dates of the file specified via **-f**. Exactly one of these methods must be used to specify the new dates. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24-hour system); the second *mm* is the minute number; *yy* is the last two digits of the year and is optional. For example:

```
settime 1008004584 ralph pete
```

sets the access and modification dates of files ralph and pete to Oct 8, 12:45 AM, 1984. Another example:

```
settime -f ralph john
```

This sets the access and modification dates of the file john to those of the file ralph.

Comments

Use of **touch**(C) in place of **settime** is encouraged.

SH(C)

Name

sh - Invokes the shell command interpreter.

Syntax

```
sh [-ceiknrstuvx] [args]
```

Description

The shell is the standard 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 0. The *value* of a simple-command is its exit status if it terminates normally, or (decimal) 1000+ status if it terminates abnormally, that is, if the failure produces a core file.

A *pipeline* is a sequence of one or more *commands* separated by a vertical bar (|). (The caret (^ has the same effect.) The standard output of each command but the last is connected by a pipe to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by ;, **&**, **&&**, or || and optionally terminated by ; or **&**. Of these four symbols, ; and **&** have equal precedence, which is lower than that of **&&** and || . The symbols **&&** and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (that is, the shell does not wait for that pipeline to finish). The symbol **&&** (||) causes the *list* following it to be executed only if the preceding pipeline returns a 0 or nonzero exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following commands. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command:

for name [in *word* **. . .] do** *list:* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* is omitted, the **for** command executes the **do** *list* once for each positional parameter that is set (see "Parameter Substitution" below). Execution ends when there are no more words in the list.

case *word* **in [***pattern***[** | *pattern***] . . .)** *list***;;] . . . esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word.* The form of the patterns is the same as that used for filename generation (see "Filename Generation" below).

If *list* **then** *list* **[elif** *list* **then** *list***] . . . [else** *list***] fi**

The *list* following **if** is executed and, if it returns a 0 exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is 0, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, the **if** command returns a 0 exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is 0, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, the **while** command returns a 0 exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list***)**

Executes *list* in a subshell.

{ *list*; **}**

The *list* is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Note: A word beginning with # causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (\` \`) may be used as part or all of a word; trailing newlines are removed.

Parameter Substitution

The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set**. Variables may be set by writing:

```
name = value [ name = value] . . .
```

Pattern-matching is not performed on *value*.

${*parameter*}

A parameter is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *, @, #, ?, –, $, and !. The value, if any, of the parameter is substituted. The braces are required only when parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A name must begin with a letter or underscore. If parameter is a digit, it is a positional parameter. If parameter is * or @, all the positional parameters, starting with **$1** are substituted (separated by spaces). Parameter **$0** is set from argument 0 when the shell is invoked.

${*parameter*:–*word*}

If parameter is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}

If parameter is not set or is null, then set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}

If parameter is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}

If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:- `pwd`}
```

If the colon (:) is omitted from the expressions on the preceding page, the shell only checks whether *parameter* is set, not whether *parameter* is null (b="").

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME	The default argument (home directory) for the **cd** command.
PATH	The search path for commands (see "Execution" below).
CDPATH	The search path for the **cd** command.
MAIL	If this variable is set to the name of a mail file, 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.

The shell gives default values to **PATH, PS1**, **PS2**, and**IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** is set by **login**(M)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters *****, **?**, and **[**. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character **.** at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly. These characters and their matching patterns are:

*****	Matches any string, including the null string.
?	Matches any single character.
[. . .]	Matches any one of the enclosed characters. A pair of characters separated by **-** matches any character lexically between the pair, inclusive. If the first character following the opening bracket (**[**) is an exclamation mark (**!**), any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > newline space tab

A character may be *quoted* (that is, made to stand for itself) by preceding it with a \ . The pair \newline is ignored. All characters enclosed between a pair of single quotation marks (′ ′), except a single quotation mark, are quoted. Inside double quotation marks (" "), parameter and command substitution occurs and \ quotes the characters \ , ` , ”, and $. The character group **”$*”** is equivalent to **”$1 $2 . . . ”**, whereas **“$@”** is equivalent to **”$1” ”$2”** . . . .

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (that is, the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple command or may precede or follow a *command*. They are not passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word*	Use file *word* as standard input (file descriptor Ø).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.
>>*word*	Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of -file); otherwise, the file is created.

<<[-]*word* The shell input is read up to a line that is the same as *word* or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\newline** is ignored, and \ must be used to quote the characters \ , **$**, ‘, and the first character of *word*. If **-** is appended to <<, all leading tabs are stripped from *word* and from the document.

<**&***digit* The standard input is duplicated from file descriptor *digit*. Similarly for the standard output using >per.

<&- The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default Ø or 1). For example:

```
 . . . 2>&1
```

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see **environ**(M)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd args
```

and

```
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, all keyword arguments are placed in the environment, even if they occur after the command name.

Signals

The Interrupt (Del) and Quit signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

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.

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and**/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A subshell (that is, 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 as specified, no input/output redirection is permitted for such commands:

:
: No effect; the command does nothing. A 0 exit code is returned.

.*file*
: Reads and executes commands from *file* and returns. The search path specified by **PATH2** is used to find the directory containing *file*.

break [*n*]
: Exits from the enclosing **for** or **while** loop, if any. If *n* is specified, breaks *n* levels.

continue [*n*]

Resumes the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resumes at the *n*-th enclosing loop.

cd [*arg*]

Changes the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is <null> (specifying the current directory). The current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*.

eval [*arg* . . .]

The arguments are read as input to the shell and the resulting commands are executed.

exec [*arg* . . .]

The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]

Causes a shell to **exit** with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file also causes the shell to exit.)

export [*name* . . .]

The given *names* are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, a list of all names that are exported in this shell is printed.

newgrp [*arg* . . .]

Equivalent to **exec newgrp** *arg* . . .

read [*name* . . .]

One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, and so on, with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* . . .]

The given *names* are marked *read-only* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *read-only* names is printed.

set [-eknuvx [*arg* **. . .]]**

-e	If the shell is noninteractive, exits immediately if a command exits with a nonzero exit status.
-k	Places all keyword arguments in the environment for a command, not just those that precede the command name.
-n	Reads commands but does not execute them.
-u	Treats unset variables as an error when substituting.
-v	Prints shell input lines as they are read.
-x	Prints commands and their arguments as they are executed.
--	Does not change any of the flags; useful in setting $1 to -.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags is in **$-**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given, the values of all names are printed.

shift [*n*]

The positional parameters from **$n+1 . . .** are renamed **$1** . . . . If *n* is not given, it is assumed to be **1**.

test

Evaluates conditional expressions. See **test**(C) for use and description.

times

Prints the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] . . .

The *arg* is a command to be read and executed when the shell receives signals *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. The highest signal number allowed is 16. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all traps *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is Ø, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

ulimit [**-f**] [*n*]

imposes a size limit of *n*.

-f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.

If no option is given, **-f** is assumed.

umask [*ooo*]

The user file-creation mask is set to the octal number *ooo* where *o* is an octal digit (see **umask**(C)). If *ooo* is omitted, the current value of the mask is printed.

wait [*n*]

Waits for the specified process to terminate and reports the termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is always 0.

Invocation

If the shell is invoked and the first character of argument 0 is **-**, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c *string*	If the **-c** flag is present, commands are read from *string*.
-s	If the **-s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.
-t	If the **-t** flag is present, a single command is read and executed and the shell exits. This flag is intended for use by C programs only and is not useful interactively.
-i	If the **-i** flag is present or if the shell input and output are attached to a terminal, this shell is interactive. In this case Terminate is ignored (so that **kill 0** does not kill an interactive shell) and Interrupt (Del) is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.
-r	If the **-r** flag is present, the shell is a restricted shell (see **rsh**(C)).

The remaining flags and arguments are described under the **set** command above.

Exit Status

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

Files

/etc/profile
$HOME/**.profile**
/tmp/sh*
/dev/null

See Also

cd(C), env(C), login(M), newgrp(C), rsh(C), test(C), umask(C), a.out(F), profile(M), environ(M)

Comments

The command **readonly** (without arguments) produces the same output as the command **export**.

If << is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a temporary file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

SHUTDOWN(C)

Name

shutdown - Terminates all processing.

Syntax

/etc/shutdown [*time*] [**su**]

Description

The **shutdown** command is part of the XENIX operating procedures. Its primary function is to terminate all currently running processes in an orderly and cautious manner. The *time* argument is the number of minutes before a shutdown will occur; default is five minutes. The optional *su* argument lets the user go single-user, without completely shutting down the system. However, the system is shut down for multiuser use. The **shutdown** command goes through the following steps. All users logged on the system are notified to log off the system by a broadcast message. All file system super-blocks are updated before the system is stopped (see **sync**(C)). This must be done before rebooting the system, to insure file system integrity.

See Also

sync(C), umount(C), wall(C)

Diagnostics

The most common error diagnostic that occurs is *device busy*. This diagnostic appears when a particular file system could not be unmounted. See **umount**(C).

Comment

Once **shutdown** has been invoked, it must be allowed to run to completion and must not be interrupted by pressing C/Break or Del.

SLEEP(C)

Name

sleep - Suspends execution for an interval.

Syntax

sleep *time*

Description

The **sleep** command 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
```

Comment

Time must be less than 65536 seconds.

SORT(C)

Name

sort - Sorts and merges files.

Syntax

```
sort [-cmubdfiınrtx] [+pos1 [-pos2] ] . . . [-ooutput] [files]
```

Description

The **sort** program merges and sorts lines from all named files and writes the result on the standard output. A dash (-) may appear as a file in the *files* argument, signifying 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 Ignores leading blanks (spaces and tabs) in field comparisons.

-d "Dictionary" order: only letters, digits and blanks are significant in comparisons.

-f Folds uppercase letters onto lowercase.

-i Ignores characters outside the ASCII octal range 040-0176 in non-numeric comparisons.

-n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b**.

-r Reverses the sequence of comparisons, that is, the output of the **sort** is in reverse order from normal.

-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* have the form *m.n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present, they override all the global ordering options for this key. If the **b** option is in effect, *n* is counted from the first non-blank in the field; **b** is attached independently to *pos2*. A missing .*n* means **.0**; a missing -*pos2* means the end of the line. Under the **-t***x* option, fields are strings separated by *x*; otherwise fields are 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. Very long lines are truncated.

These option arguments are also understood:

-c   Checks that the input file is sorted according to the ordering rules; gives no output unless the file is out of sort.

-m   Merges only, the input files are already sorted.

-u   Suppresses all but one instance in each set of duplicated lines. Ignored bytes and bytes outside keys do not participate in this comparison.

-o   The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

Examples

The following prints in alphabetical order all the unique spellings in a list of words (capitalized words differ from uncapitalized):

```
sort -u +Øf +Ø list
```

The following prints the password file (**passwd**(M)) sorted by user ID (the third colon-separated field):

```
sort -t: +2n /etc/passwd
```

The following prints 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 +Ø -1 dates
```

File

/usr/tmp/stm???

See Also

comm(C), join(C), uniq(C)

Diagnostic

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **-c**.

SPLIT(C)

Name

split - Splits a file into pieces.

Syntax

split [-*n*] [*file*[*name*]]

Description

The **split** command 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 the default.

If no input file is given, or if a dash (**-**) is given instead, the standard input file is used.

See Also

bfs(C), csplit(C)

STTY(C)

Name

stty - Sets the options for a terminal.

Syntax

stty [**-a**] [**-g**] [*options*]

Description

The **stty** command sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **-a** option, it reports all of the option settings; with the **-g** option, it reports current settings in a form that can be used as an argument to another **stty** command. Detailed information about the modes listed in the first five groups below may be found in **tty**(M). Options in the last group are implemented using options in the previous groups. The options are selected from the following:

Control Modes

parenb (-parenb)
Enables (disables) parity generation and detection.

parodd (-parodd)
Selects odd (even) parity.

cs5 cs6 cs7 cs8
Selects character size (see **tty**(M)).

0
Hangs up phone line immediately.

50 75 110 134 150 200 300 600
1200 1800 2400 4800 9600 exta

Sets terminal baud rate to the number given, if possible. The baud rate that is possible is defined in the appropriate Hardware Reference Manual.

hupcl (-hupcl)

Hangs up (does not hang up) phone connection on last close.

hup (-hup)

Same as hupcl (-hupcl).

cstopb (-cstopb)

Uses two(one) stop bits per character.

cread (-cread)

Enables (disables) the receiver.

clocal (-clocal)

Assumes a line without (with) modem control.

Input Modes

ignbrk (-ignbrk)

Ignores (does not ignore) break on input.

brkint (-brkint)

Signals (does not signal) Intr on break.

ignpar (-ignpar)

Ignores (does not ignore) parity errors.

parmrk (-parmrk)

Marks (does not mark) parity errors (see **tty(M)**).

inpck (-inpck)

Enables (disables) input parity checking.

istrip (-istrip)

Strips (does not strip) input characters to 7 bits.

inlcr (-inlcr)
Maps (does not map) NL to CR on input.

igncr (-igncr)
Ignores (does not ignore) CR on input.

icrnl (-icrnl)
Maps (does not map) CR to NL on input.

iuclc (-iuclc)
Maps (does not map) uppercase alphabetics to lowercase on input.

ixon (-ixon)
Enables (disables) start/stop output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (-ixany)
Allows any character (only DC1) to restart output.

ixoff (-ixoff)
Requests that the system send (not send) Start/Stop characters when the input queue is nearly empty/full.

Output Modes

opost (-opost)
Post-processes output (does not post-process output; ignores all other output modes).

olcuc (-olcuc)
Maps (does not map) lowercase alphabetics to uppercase on output.

onlcr (-onlcr)
Maps (does not map) NL to CR-NL on output.

ocrnl (-ocrnl)
Maps (does not map) CR to NL on output.

onocr (-onocr)
Does not (does) output CRs at column zero.

onlret (-onlret)
On the terminal NL performs (does not perform) the CR function.

ofill (-ofill)
Uses fill characters (use timing) for delays.

ofdel (-ofdel)
Fill characters are Dels (Nuls)

cr0 cr1 cr2 cr3.
Selects style of delay for carriage returns (see **tty**(M)).

nl0 nl1
Selects style of delay for linefeeds (see **tty**(M)).

tab0 tab1 tab2 tab3
Selects style of delay for horizontal tabs (see **tty**(M)).

bs0 bs1
Selects style of delay for backspaces (see **tty**(M)).

ff0 ff1
Selects style of delay for form feeds (see **tty**(M)).

vt0 vt1
Selects style of delay for vertical tabs (see **tty**(M)).

Local Modes

isig (-isig)
Enables (disables) the checking of characters against the special control characters Intr and Quit.

icanon (-icanon)
Enables (disables) canonical input (erase and kill processing).

xcase (-xcase)
Canonical (unprocessed) upper/lowercase presentation.

echo (-echo)
Echoes back (does not echo back) every character typed.

echoe (-echoe)
Echoes (does not echo) Erase character as a backspace-space-backspace string.

> **Note:** This mode erases the ERASEed character on many displays; however, it does not keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (-echok)
Echoes (does not echo) NL after KILL character.

lfkc (-lfkc)
The same as **echok (-echok)**

echonl (-echonl)
Echoes (does not echo) NL.

noflsh (-noflsh)
Disables (enables) flush after Intr or Quit.

Control Assignments

control-character-C
Sets *control-character* to *C*, where *control-character* is **erase, kill, intr, quit, eof, eol**. If *C* is preceded by a caret (∧) (escaped from the shell), the value used is the corresponding Ctrl character (for example, "∧D" is a Ctrl-D); "∧?" is interpreted as Del and "∧-" is interpreted as undefined.

min **i**, *time* **i(0<i<127)**
When **icanon** is not set, read requests are not satisfied until at least *min* characters have been received or the timeout value *time* has expired. See **tty(M)**.

line i

Sets the line discipline to $i(0 < i < 127)$. There are currently no line disciplines implemented.

Combination Modes

evenp or parity

Enables **parenb** and **cs7**.

oddp

Enables **parenb**, **cs7**, and **parodd**.

-parity, -evenp, or -oddp

Disables **parenb**, and sets **cs8**.

raw (-raw or cooked)

Enables (disables) raw input and output (no Erase, Kill, Intr, Quit, top, or output postprocessing).

nl (-nl)

Unsets (sets) **icrnl**, **onlcr**. In addition, **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (-lcase)

Sets (unsets) **xcase**, **iuclc**, and **olcuc**.

LCASE (-LCASE)

Same as **lcase** (**-lcase**).

tabs (tabs or tab3)

Preserves (expands to spaces) tabs when printing.

ek

Resets Erase and Kill characters back to normal Ctrl-H and Ctrl-U.

sane

Resets all modes to some reasonable values. Useful when a terminal's settings have been scrambled.

term

Sets all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

See Also

tty(M)

Comment

Many combinations of options make no sense, but no checking is performed.

SU(C)

Name

su - Makes the user super-user or another user.

Syntax

su [**-**] [*name* [*arg* . . .]]

Description

The **su** command allows you to become another user without logging off. The default user *name* is **root** (that is, super-user).

To use **su**, the appropriate password must be supplied (unless you are already super-user). If the password is correct, **su** executes a new shell with the user ID set to that of the specified user. To restore normal user ID privileges, type a Ctrl-D to the new shell.

Any additional arguments are passed to the shell, permitting the super-user to run shell procedures with restricted privileges (an *arg* of the form **-c** *string* executes *string* via the shell). When additional arguments are passed, **/bin/sh** is always used. When no additional arguments are passed, **su** uses the shell specified in the password file.

An initial dash (**-**) causes the environment to be changed to the one that would be expected if the user actually logged in again. This is done by invoking the shell with an *arg∅* of **-su** causing the **.profile** in the home directory of the new user ID to be executed. Otherwise, the environment is passed along with the possible exception of $PATH, which is set to **/bin:/etc:/usr/bin** for root.

Note that **.profile** can check *arg∅* for **-sh** or **-su** to determine how it was invoked. This is true only if there is no specified shell (in the **passwd** file) for the user. If a shell has been specified in the **passwd** file, the "shell name" is passed to *arg∅*.

Files

/etc/passwd	The system password file
$HOME/.profile	User's profile

See Also

env(C), login(M), sh(C), environ(M)

SUM(C)

Name

sum - Calculates checksum and counts blocks in a file.

Syntax

sum [**-r**] *file*

Description

The **sum** command 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 data or to validate a file communicated over a transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

See Also

wc(C)

Diagnostics

"Read error" is indistinguishable from end-of-file on most devices; check the block count.

SYNC(C)

Name

sync - Updates the super-block.

Syntax

sync

Description

The **sync** command executes the **sync** system primitive. If the system is to be stopped, **sync** must be called to ensure file system integrity. Note that **shutdown**(C) automatically calls **sync** before shutting down the system.

See Also

shutdown(C)

SYSADMIN(C)

Name

sysadmin - Performs file system backups and restores files.

Syntax

```
/etc/sysadmin
```

Description

The **sysadmin** script performs file system backups and restores to and from backup disks. It can do a daily incremental backup (level 9), or a periodic full backup (level Ø). It can provide a listing of the files backed up and also has a facility to restore individual files from a backup.

The **sysadmin** script operates on XENIX formatted diskettes. The version provided backs up the root and user file systems.

The script can be edited to operate on additional file systems if required.

You must be the super-user to use this program.

File

/tmp/backup.list

See Also

backup(C), restor(C), mkfs(C), dumpdir(C)

TAIL(C)

Name

tail - Copies the last part of a file to the output.

Syntax

```
tail [ ±[number] [lbc] [-f] ] [file]
```

Description

The **tail** command copies the named file to the standard output beginning at a designated place. If no *file* is named, the standard input is used.

Copying begins at distance +*number* from the beginning, or -*number* from the end of the input (if *number* is null, the value 10 is assumed). The *number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **-f** (follow) option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus, it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f file
```

prints the last ten lines of **file**, followed by any lines that are appended to **file** between the time **tail** is initiated and ended.

See Also

dd(C)

Comment

Any **tail** commands relative to the end of the file are kept in a buffer and thus are limited in length. Unpredictable results can occur if character special files are "tailed."

TAR(C)

Name

tar - Archives files.

Syntax

tar [*key*] [*files*]

Description

The **tar** command saves and restores files to and from an archive medium which is typically a storage device such as diskette, tape, or regular file. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Valid function letters are **r**, **x**, **t**, **u**, and **c**.. Other arguments to the command are *files* (or directory names) specifying which files are to be backed up or restored. In all cases, appearance of a directory name refers to the files and (recursively) to subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named *files* are written to the end of the archive. The **c** function implies this function.

x The named *files* are extracted from the archive. If a named file matches a directory whose contents had been written onto the archive, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire contents of the archive are extracted. Note that if several files with the same name are on the archive, the last one overwrites all earlier ones.

t The names of the specified files are listed each time that they occur on the archive. If no *files* argument is given, all the names on the archive are listed.

u The named *files* are added to the archive if they are not already there or if they have been modified since last written on that archive.

c Creates a new archive; writing begins at the beginning of the archive, instead of after the last file. This command implies the **r** function.

The following characters may be used in addition to the letter that selects the desired function:

e Prevents files from being split across volumes (tapes or diskettes). If there is not enough room on the present volume for a given file, **tar** prompts for a new volume. This is only valid when the **-k** option is also specified on the command line.

0, . . . ,7

This modifier selects the drive on which the archive is mounted. The default is **1[for/dev/m+1]**. This option should only be selected if you have linked the appropriate /**dev/mt** to the desired device.

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 archive entries than just the name.

w Causes **tar** to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no."

f Causes **tar** to use the next argument as the name of the archive instead of **/dev/mt1**. If the name of the file is a dash (-), **tar** writes to the standard output or reads from the standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a pipeline. The **tar** command can also be used to move hierarchies with the command:

```
cd fromdir; tar cf - .| (cd todir; tar xf -)
```

b	Causes **tar** to use the next argument as the blocking factor for archive records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).
F	Causes **tar** to use the next argument as the name of a file from which succeeding arguments are taken. A lone dash (**-**) signifies that arguments are taken from the standard input.
l	Tells **tar** to print an error message if it cannot resolve all of the links to the files being backed up. If **l** is not specified, no error messages are printed.
m	Tells **tar** to not restore the modification times. The modification time of the file is the time of extraction.
k	Causes **tar** to use the next argument as the size of an archive volume in kilobytes. The minimum value allowed is 250. This option is useful when the archive is not intended for a magnetic tape device, but for some fixed size device, such as diskette (See **f** above). Very large files are split into "extents" across volumes. When restoring from a multivolume archive, **tar** only prompts for a new volume if a split file has been partially restored.
n	Indicates the archive device is not a magnetic tape. The **k** option implies this. Listing and extracting the contents of an archive are sped because **tar** can seek over files it wishes to skip. Sizes are printed in kilobytes instead of tape blocks.
p	Indicates that files are extracted using their original permissions. It is possible that a non-super-user may be unable to extract files because of the permissions associated with the files or directories being extracted.

Examples

If the name of a diskette device is **/dev/fdØ96ds15**, a file can be copied in tar format on this device by typing:

```
tar cvfk /dev/fdØ96ds15 1150 files
```

where *files* are the names of files you want archived and 1150 is the capacity of the diskette in kilobytes. Arguments to key letters are given in the same order as the key letters themselves, thus the **fk** key letters have corresponding arguments **/dev/fdØ96ds15**and **1150**. If a *file* is a directory, the contents of the directory are recursively archived. To print a listing of the archive, type:

```
tar tvf /dev/fdØ96ds15
```

At some later time you will likely want to extract the files from the archive diskette. You can do this by typing:

```
tar xvf /dev/fdØ96ds15
```

The above command extracts all files from the archive using the exact same path names as used when the archive was created. Because of this behavior, it is normally best to save archive files with relative path names rather than absolute ones, because directory permissions may not let you read the files into the absolute directories specified.

In the above examples, the **v** (verbose) option is used simply to confirm the reading or writing of archive files on the screen. Also, a normal file could be substituted for the diskette drive **/dev/fdØ96ds15**in the examples.

Files

/etc/default/backup	Default devices
/tmp/tar*	

Diagnostics

Prints an error message about bad key characters and archive read/write errors.

Prints an error message if not enough memory is available to hold the link tables.

Comments

There is no way to ask for the *n*th occurrence of a file.

The **u** option can be slow.

The **b** option should not be used with archives that are going to be updated. If the archive is on a disk, the **b** option should not be used at all, because updating an archive stored on disk can destroy it. To update (**r** or **u** option) a tar archive, do not use raw magnetic tape and do not use the **b** option. This applies both when updating and when the archive was first created.

The limit on filename length is 100 characters.

When archiving a directory that contains subdirectories, **tar** can only access those subdirectories that are within 17 levels of nesting. Subdirectories at higher levels will be ignored after **tar** displays an error message.

Systems with a 1K-byte file system cannot specify raw disk devices unless the **b** option is used to specify an even number of blocks. This means that one cannot update a raw-mode disk partition.

Don't do:

```
tar xfF - -
```

This would imply taking two things from the standard input at the same time.

TEE(C)

Name

tee - Creates a tee in a pipe.

Syntax

tee [-i] [-a] [*files*] . . .

Description

The **tee** command transcribes the standard input to the standard output and makes copies in the *files*. The **-i** option ignores interrupts; the **-a** option causes the output to be appended to the *files* rather than overwriting them.

Examples

The following example illustrates the creation of temporary files at each stage in a pipeline:

```
grep ABC | tee ABC.grep | sort | tee ABC.sort | more
```

This example shows how to tee output to the terminal screen:

```
grep ABC | tee /dev/tty | sort | uniq >final.file
```

TEST(C)

Name

test - Tests conditions.

Syntax

```
test expr
[expr]
```

Warning: In the second form of the command (that is, the one that uses **[ ]**, rather than the word **test**), the square brackets must be delimited by blanks.

Description

The **test** command evaluates the expression *expr* and, if its value is true, returns a 0 (true) exit status; otherwise, returns a 1 (false) exit status. The **test** command returns a nonzero exit status if there are no arguments. The following primitives are used to construct *expr*:

-r*file*	True if *file* exists and is readable.
-w*file*	True if *file* exists and is writable.
-x*file*	True if *file* exists and is executable.
-f*file*	True if *file* exists and is a regular file.
-d*file*	True if *file* exists and is a directory.
-c*file*	True if *file* exists and is a character special file.
-b*file*	True if *file* exists and is a block special file.
-u*file*	True if *file* exists and its set-user-ID bit is set.
-g*file*	True if *file* exists and its set-group-ID bit is set.

-k*file*	True if *file* exists and its sticky bit is set.
-s*file*	True if *file* exists and has a size greater than zero.
-t[*fildes***]**	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
-z *s1*	True if the length of string *s1* is zero.
-n *s1*	True if the length of the string *s1* is nonzero.
s1 = *s2*	True if strings *s1* and *s2* are identical.
s1 != *s2*	True if strings *s1* and *s2* are **not** identical.
s1	True if *s1* is **not** the null string.
n1 **-eq** *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

!	Unary negation operator
-a	Binary AND operator
-o	Binary OR operator (**-a** has higher precedence than **-o**)
(expr)	Parentheses for grouping

Notice that all the operators and flags are separate arguments to **test**. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

See Also

find(C), sh(C)

TOUCH(C)

Name

touch - Updates access and modification times of a file.

Syntax

touch [**-amc**] [*mmddhhmm*[*yy*]] *files*

Description

The **touch** command causes the access and modification times of each argument to be updated. If no time is specified (see **date**(C), the current time is used. The **-a** and **-m** options cause touch to update only the access or modification times respectively (default is **-am**). The **-c** option silently prevents **touch** from creating the file if it did not previously exist.

The return code from **touch** is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

See Also

date(C)

TR(C)

Name

tr - Translates characters.

Syntax

tr [**-cds**] [*string1* [*string2*]]

Description

The **tr** command copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **-cds** may be used:

-c	Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
-d	Deletes all input characters in *string1*.
-s	Compresses all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a-z]	Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.
[a*n**]**	Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A 0 or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by one, two, or three octal digits stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in *file1*, one per line, in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline:

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

See Also

ed(C), sh(C), ascii **(M)**

Comment

The **tr** command won't handle ASCII Nul in *string1* or *string2*; it always deletes Nul from input.

TRUE(C)

Name

true - Returns with a 0 exit value.

Syntax

true

Description

The **true** command does nothing except return with a 0 exit value. The **false** command, **true**'s counterpart, does nothing except return with a nonzero exit value. The **true** command is typically used in shell procedures such as:

```
while true
do
        command
done
```

See Also

sh(C), false(C)

Diagnostics

The **true** command has exit status 0.

TSET(C)

Name

tset - Sets terminal modes.

Syntax

tset [**-**] [**-hrsIQS**] [**-e**[*c*]] [**-E**[*c*]] [**-k**[*c*]]
[**-m**[*ident*][*test baudrate*]:*type*] [*type*]

Description

The **tset** command allows the user to set a terminal's Erase and Kill characters, and define the terminal's type and capabilities by creating values for the TERM and TERMCAP environment variables. If a *type* is given, **tset** creates information for a terminal of the given type. The type may be any type given in **/etc/termcap**. If *type* is not given, **tset** creates information for a terminal of the type defined by the value of the environment variable TERM, unless the **-h** or **-m** option is given. If these options are given, **tset** searches the **/etc/ttytype** file for the terminal type corresponding to the current serial port, then creates information for a terminal based on this type. If the serial port is not found in **/etc/ttytype** the terminal type is set to *unknown.*

The *tset* command displays the created information at the standard output. The information is in a form that can be used to set the current environment variables. The exact form depends on the login shell from which **tset** was invoked. Examples below illustrate how to use this information to change the variables.

There are the following options:

-	Prints the terminal type on the standard output.
-r	Prints the terminal type on the diagnostic output.

-s Outputs **export** and assignment commands (for **sh**(C)). The *type* of commands is determined by the user's login shell.

-I Suppresses printing of the terminal initialization strings.

-Q Suppresses the printing of the "Erase set to" and "Kill set to" messages.

-S Only outputs the strings to be placed in the environment variables.

-e[*c*] Sets the Erase character to Ctrl-*C* on all terminals. The default for *c* is the backspace character on the terminal, usually Ctrl-H.

-E[*c*] Identical to the **-e** command except that it only operates on terminals that can backspace.

-k[*c*] Sets the Kill character to Ctrl-*C*, defaulting to Ctrl-U.

-m[*ident*][*test baudrate*]:*type*

Allows a user to specify how a given serial port is is to be mapped to an actual terminal type. The option applies to any serial port in **/etc/ttytype** whose type is indeterminate (for example, *dialup*, *plugboard*, and so on. The *type* specifies the terminal type to be used, and *ident* identifies the name of the indeterminate type to be matched. If no *ident* is given, all indeterminate types are matched. The *test baudrate* defines a test to be performed on the serial port before the type is assigned. The *baudrate* must be as defined in **stty**(C). The *test* may be any combination of: **> = < @ and !**. If the *type* begins with a question mark, the user is asked if he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. The question mark must be escaped to prevent filename expansion by the shell. If more than one **-m** option is given, the first correct mapping prevails.

The *tset* command is most useful when included in the **.profile** (for **sh**(C)) file executed automatically at login, with **-m** mapping used to specify the terminal type you most frequently dial in on.

Examples

```
tset gt42

tset -m dialup\>300:adm3a -m dialup:dw2 -Qr -e#

tset -m dial:ti733 -m plug:\?hp2621 -m unknown:\? -e -k^U
```

To use the information created by the **-s** option for the Bourne shell (**sh**), repeat these commands:

```
tset -s  . . .  > /tmp/tset$$
/tmp/tset$$
rm /tmp/tset$$
```

Files

/etc/ttytype	Port name to terminal type map database
/etc/termcap	Terminal capability database

See Also

tty(M), termcap(M), stty(C)

Credit

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

TTY(C)

Name

tty - Gets the terminal's name.

Syntax

```
tty [-s]
```

Description

The tty command prints the path name of the user's terminal on the standard output. The **-s** option inhibits printing, allowing you to test just the exit code.

Exit Codes

0; if the standard input is a terminal, 1 otherwise.

Diagnostic

not a tty	If the standard input is not a terminal and **-s** is not specified.

UMASK(C)

Name

umask - Sets file-creation mode mask.

Syntax

umask [*ooo*]

Description

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for owner, group, and others, respectively. Only the low-order nine bits of **umask** and the file mode creation mask are used. The value of each specified digit is subtracted from the corresponding digit specified by the system for the creation of any file. This is actually a binary masking operation, and thus the name **umask**. In general, binary 1's remove a given permission and 0's have no effect. For example, **umask 022** removes group and others write permission (files normally created with mode 777 become mode 755; files created with mode 666 become mode 644).

If *ooo* is omitted, the current value of the mask is printed.

The **umask** command is recognized and executed by the shell. By default, login shells have a umask of 022.

See Also

chmod(C), sh(C)

UMOUNT(C)

Name

umount - Dismounts a file structure.

Syntax

/etc/umount *special-device*

Description

The **umount** command announces to the system that the removable file structure previously mounted on device *special-device* is to be removed. Any pending I/O for the file system is completed, and the file structure is flagged clean. For fuller explanation of the mounting process, see **mount**(C).

Files

/etc/mnttab	Mount table

See Also

mount(C), mnttab(F)

Diagnostic

device busy	An executing process is using a file on the named file system.

UNAME(C)

Name

uname - Prints the current XENIX name.

Syntax

uname [-snrmduva]

Description

The **uname** command prints the current system name of XENIX on the standard output file. The options cause selected information returned by **name** to be printed:

-s Prints the system name (default).

-n Prints the nodename (the nodename may be a name that the system is known by to a communications network).

-r Prints the operating system release.

-m Prints manufacturer, original supplier of XENIX system.

-d Prints distributor or OEM for this system.

-u Prints user serial number for this system.

-v Prints the operating system version.

-a Prints all the above information.

UNIQ(C)

Name

uniq - Reports repeated lines in a file.

Syntax

uniq [**-udc**[+*n*] [-*n*]] [*input*[*output*]]

Description

The **uniq** command reads the *input* file and compares adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the *output* file. *Input* and *output* should always be different. Repeated lines must be adjacent to be found (see **sort**(C)). If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-*n* The first *n* fields and any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

See Also

comm(C), sort(C)

UNITS(C)

Name

units - Converts units.

Syntax

```
units
```

Description

The **units** command 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

                 * 2.54ØØØØe+ØØ

                 / 3.937ØØ8e-Ø1
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have:  15 lbs force/in2
You want:  atm

                 * 1.Ø2Ø689e+ØØ

                 / 9.797299e-Ø1
```

The **units** command only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, as well as the following:

pi	Ratio of circumference to diameter
c	Speed of light
e	Charge on an electron
g	Acceleration of gravity
force	Same as **g**
mole	Avogadro's number
water	Pressure head per unit height of water
au	Astronomical unit

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (for example **lightyear**). British units that differ from their US counterparts are prefixed with **br**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

File

/usr/lib/unittab

UUCLEAN(C)

Name

uuclean - Clean up the uucp spool directory.

Syntax

/usr/lib/uucp/uuclean [*options*]

Description

The **uuclean** command scans the spool directory for files with the specified prefix and deletes all those that are older than the specified number of hours.

The following options are available.

-d*directory*
Clean *directory* instead of the spool directory.

-p*pre* Scan for files with *pre* as the file prefix. Up to 10 **-p** arguments may be specified. A **-p** without any *pre* following will cause all files older than the specified time to be deleted.

-n*time* Files whose age is more than *time* hours are deleted if the prefix test is satisfied. (default time is 72 hours)

-m Send mail to the owner of the file when it is deleted.

This program is typically started by **cron**(C).

Files

/usr/lib/uucp	directory where **uuclean** command is located
/usr/spool/uucp	spool directory

See Also

uucp(C), uux(C).

UUCP(C)

Name

uucp, uulog - Copies files from XENIX to XENIX

Syntax

```
uucp [option] ... source-file ... destination-file

uulog [option]
```

Description

The **uucp** command copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on your system, or may have the form:

system-name!*pathname*

where "*system-name*" is taken from a list of system names that **uucp** knows about. Shell metacharacters ?*[] appearing in *pathname* are expanded on the appropriate system.

Pathnames may be a a full pathname, or a pathname preceded by ~ *user* where *user* is a user ID on the specified system and is replaced by that user's login directory. 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-filename* is used.

The **uucp** command preserves execute permissions across the transmission and gives 0666 read and write permissions.

The following options are interpreted by **uucp** :

-d Makes all necessary directories for the file copy.

-c Uses the source file when copying out rather than copying the file to the spool directory.

-m Sends mail to the requester when the copy is complete.

The **uulog** command maintains a summary log of **uucp** and **uux (C)** 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:

-s*sys*
Prints information about work involving system *sys*.

-u*user*
Prints 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(C), mail(C)

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.

Comments

For security reasons, all files received by **uucp** should 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.

This version of **uucp** is based on a version 7 implementation of the program.

UUSTAT(C)

Name

uustat - uucp status inquiry and job control.

Syntax

uustat [*option*] . . .

Description

The **uustat** command will display the status of, or cancel, previously specified uucp commands, or provide general status on uucp connections to other systems. The following options are recognized:

-m*mch*	Report the status of accessibility of machine *mch* . If *mch* is specified as **all,** the status of all machines known to the local uucp is provided.
-k*jobn*	Kill the uucp request whose job number is *jobn.* Jobn can be found by using the **-u** option. The killed uucp request must belong to the person issuing the **uustat** command unless that person is the super-user.
-c*hour*	Remove the status entries that are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.
-u*user*	Report the status of all uucp requests issued by *user.*
-s*sys*	Report the status of all uucp requests that communicate with remote system *sys.*
-o*hour*	Report the status of all uucp requests that are older than *hour* hours.
-y*hour*	Report the status of all uucp requests that are younger than *hour* hours.

-j*all*	Report the status of all the uucp requests.
-v	Report the uucp status verbosely. If this option is not specified, a status code is printed with each uucp request.

When no options are given, **uustat** outputs the status of all uucp requests issued by the current user. Note that you can only specify one of the options: **-j** , **-m** , **-k** , or **-c** at a time.

For example, the command

```
uustat -uhdc -smhtsa -y72 -v
```

prints the verbose status of all uucp requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The job request status format is:

```
job-number user remote-system command-time status-time
```

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	**STATUS**
00001	The copy failed, but the reason cannot be determined
00002	Permission to access local file is denied
00004	Permission to access remote file is denied
00010	Bad uucp command is generated
00020	Remote system cannot create temporary file
00040	Cannot copy to remote directory
00100	Cannot copy to local directory
00200	Local system cannot create temporary file
00400	Cannot execute uucp
01000	Copy succeeded
02000	Copy finished, job deleted
04000	Job is queued

The machine accessibility status format is:

```
system-name time status
```

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

Files

/usr/spool/uucp	spool directory
/usr/lib/uucp/L__stat	system status file
/usr/lib/uucp/R__stat	request status file

See Also

uucp(C).

UUSUB(C)

Name

uusub - Monitor uucp network.

Syntax

uusub [*options*]

Description

The **uusub** command defines a uucp subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

-a*sys*	Add *sys* to the subnetwork.
-d*sys*	Delete *sys* from the subnetwork. (super user only)
-l	Report the statistics on connections.
-r	Report the statistics on traffic amount.
-f	Flush the connection statistics.
-u*hr*	Gather the traffic statistics over the past *hr* hours.
-c*sys*	Exercise the connection to the system *sys*. If *sys* is specified as **all,** exercise the connection to all the systems in the subnetwork.

Sys in the above options is the first seven characters of a system name.

The connections report format is:

```
sys #call #ok time #dev #login #nack #other
```

Where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, *#ok* is the number of successful connections, *time* is the the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (for example ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (for example line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The traffic statistics format is:

```
sfile sbyte rfile rbyte
```

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest **uusub** command with the **-u***hr* option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

The command:

```
uusub -c all -u 24
```

is typically started by **cron(C)** once a day.

Files

/usr/spool/uucp/SYSLOG	system log file
/usr/lib/uucp/L__sub	connection statistics
/usr/lib/uucp/R__sub	traffic statistics

See Also

uucp(C), uustat(C).

UUTO(C)

Name

uuto, uupick - Public XENIX-to-XENIX file copy.

Syntax

```
uuto [options] source-files destination
uupick[-ssystem]
```

Description

The **uuto** command sends *source-files* to *destination.* The **uuto** command uses the **uucp(C)** facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that uucp knows about (see uuname(C)). *User* is the login name of someone on the specified system.

Two options are available:

-p Copy the source file into the spool directory before transmission.

-m Send mail to the sender when the copy is complete.

The files (or subtrees if directories are specified) are sent to PUBDIR on *system* , where PUBDIR is a public directory defined in the uucp source. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by **mail**(C) of the arrival of files.

The **uupick** command accepts or rejects the files transmitted to the user. Specifically, **uupick** searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*:[file *file-name*] [dir *dirname*]?

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m[*dir*]	Move the entry to named directory *dir* (current directory is default).
a[*system*]	Same as **m** except moving all the files sent from *system* .
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q** .
! *command*	Escape to the shell to do *command* .
*****	Print a command summary.

Uupick invoked with the **-s** *system* option will only search the PUBDIR for files sent from *system* .

Files

PUBDIR/usr/spool/uucppublic   public directory

See Also

mail(C), uuclean(C), uucp(C), uuname(C), uustat(C), uux(C).

UUX(C)

Name

uux - Executes commands on remote XENIX systems

Syntax

uux [-] *command-string*

Description

The **uux** command will gather 0 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 filenames may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

Filenames may be (1) a full pathname; (2) a pathname preceded by *~xxx ;* where *xxx* is a user ID on the specified system and is replaced by that user's login directory; or (3) anything else prefixed by the current directory.

The "-" option causes 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 systems, 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/spool/uucp	Spool directory
/usr/lib/uucp/*	Other data and programs

See Also

uucp(C)

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.**

Comments

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 shell metacharacter * will probably not perform as expected.

The shell tokens << and >> are not implemented.

There is no notification of denial of execution on the remote machine.

The *uux* command does not execute a command on a remote system unless the remote system has the name of the command listed in the */usr/lib/uucp/L.cmds* file.

VI(C)

Name

vi - Invokes a screen-oriented display editor.

Syntax

```
vi [-option ...] [command] [filename]
```

Description

The **vi** editor offers a powerful set of text editing operations based on a set of mnemonic commands. Most commands are single keystrokes that perform simple editing functions. This editor displays a full screen "window" into the file you are editing. The contents of this window can be changed quickly and easily within vi. While editing, visual feedback is provided (the name **vi** is short for "visual").

The **vi** editor and the line editor **ex** are one and the same editor: the names **vi** and **ex** identify a particular user interface rather than any underlying functional difference. The differences in user interface, however, are quite striking; **ex** is a powerful line-oriented editor, similar to the editor **ed**. However, in both **ex** and **ed**, visual updating of the terminal is limited, and commands are entered on a command line. On the other hand, **vi** is a screen-oriented editor designed so that what you see on the screen corresponds exactly and immediately to the contents of the file you are editing.

Options available on the **vi** command line:

-t Equivalent to an initial *tag* command; edits the file containing the tag and positions the editor at its definition.

-r Used in recovering after an editor or system failure, retrieving the last saved version of the named file. If no file is specified, this option prints a list of saved files.

-l Specific to editing LISP, this option sets the *showmatch* and *lisp* options.

-w*n* Sets the default window size to *n*. Useful on dialups to start in small windows.

-R Sets a read-only option so that files can be viewed but not edited.

The Editing Buffer

The **vi** editor performs no editing operations on the file that you name during invocation. Instead, it works on a copy of the file in an editing buffer. The editor remembers the name of the file specified at invocation so that it can later copy the editing buffer back to the named file. The contents of the named file are not affected until the changes are copied back to the original file. This allows editing of the buffer without immediately destroying the contents of the original file.

When you invoke **vi** with a single filename argument, the named file is copied to a temporary editing buffer. When the file is written out, the temporary file is written back to the named file.

Modes of Operation

Within **vi** there are three distinct modes of operation:

Command Mode

Within command mode, signals from the keyboard are interpreted as editing commands.

Insert Mode

Insert mode can be entered by typing any of the **vi** insert, append, open, substitute, change, or replace commands. Once in insert mode, letters typed at the keyboard are inserted into the editing buffer.

Ex Escape Mode

The **vi** and **ex** editors are one and the same editor differing mainly in their user interface. In **vi**, commands are usually single keystrokes. In **ex**, commands are lines of text terminated by an Enter. A special "escape" command gives **vi** access to many of these line-oriented **ex** commands. To escape to **ex** escape mode, type a colon (:). The colon is echoed on the status line as a prompt for the **ex** command. An executing command can be aborted by pressing Interrupt (Del). Most file manipulation commands are executed in **ex** escape mode; for example, the commands to read in a file and to write out the editing buffer to a file.

Special Keys

There are several special keys in **vi**. These keys are used to edit, delimit, or abort commands and command lines.

Esc Used to return to **vi** command mode, cancel partially formed commands.

Enter
Terminates **ex** commands when in **ex** escape mode. Also used to start a new line when in insert mode.

Interrupt (Del)
Often the same as the Del or Erase key on many terminals. Generates an interrupt, telling the editor to stop what it is doing. Used to end any command that is executing.

/ Used to specify a string to be searched for. The slash appears on the status line as a prompt for a search string.

? Works exactly like the slash key, except that it is used to search backward in a file instead of forward.

:	The colon is a prompt for an **ex** command. You can then type in any **ex** command, followed by an Esc or Enter. and the given ex command is executed.

The following characters are special in insert mode:

Bksp	Backs up the cursor one character on the current line. The last character typed before the Bksp is removed from the input buffer but remains displayed on the screen.
Ctrl-U	Moves the cursor back to the first character of the insertion, and restarts insertion.
Ctrl-V	Removes the special significance of the next typed character. Use Ctrl-V to insert control characters. Line feed and Ctrl-J cannot be inserted in the text except as newline characters. Ctrl-Q and Ctrl-S are trapped by the operating system before they are interpreted by **vi,** so they too cannot be inserted as text.
Ctrl-W	Moves the cursor back to the first character of the last inserted word.
Ctrl-T	During an insertion, with the *autoindent* option set and at the beginning of the current line, typing this character inserts *shiftwidth* whitespace.
Ctrl-@	If typed as the first character of an insertion, it is replaced with the last text inserted, and the insertion terminates. Only 128 characters are saved from the last insertion. If more than 128 characters were inserted, this command inserts no characters. A Ctrl-@ cannot be part of a file, even if quoted.

Invoking and Exiting Vi

To enter **vi** type:

vi	Edits empty editing buffer
vi *file*	Edits named file
vi +123*file*	Goes to line 123
vi +45 *file*	Goes to line 45
vi +/ *word file*	Finds first occurrence of "word"
vi +/*tty file*	Finds first occurrence of "tty"

There are several ways to exit the editor:

zz	The editing buffer is written to the file *only* if changes were made.
:x	The editing buffer is written to the file *only* if changes were made.
:q!	Cancels an editing session. The exclamation mark (!) tells **vi** to quit unconditionally. In this case, the editing buffer is not written out.

Vi Commands

The **vi** editor is a visual editor that displays a window of the file. What you see on the screen is **vi**'s notion of what the file contains. Commands do not cause any change to the screen until the complete command is typed. Most commands may take a preceding *count* that specifies repetition of the command. This count parameter is not given in the following command descriptions but is implied unless overridden by some other prefix argument. When **vi** gets an improperly formatted command, it sounds a beep.

Cursor Movement

The cursor movement keys allow you to move your cursor around in a file. Note in particular the arrow keys (if available on your terminal), the h, j, k, and l cursor keys, and Space, Bksp, Ctrl-N, and Ctrl-P. These three sets of keys perform identical functions.

Forward Space: -l, Space, or -->

Syntax: **l**
Space
-->

Function: Moves the cursor right one character. If a count is given, moves right count characters. You cannot move past the end of the line.

Backspace: -h, Bksp, or <--

Syntax: **h**
Bksp
<--

Function: Moves cursor left one character. If a count is given, moves left *count* characters. You cannot move past the beginning of the current line.

Next Line: -+, Enter, j, Ctrl-N, and LF

Syntax: +
Enter

Function: Moves the cursor down to the beginning of the next line.

Syntax: **j**
Ctrl-N
LF
(down arrow)

Function: Moves the cursor down one line, remaining in the same column. Note the difference between these commands and the preceding set of next line commands, which move to the *beginning* of the next line.

Previous Line: -k, Ctrl-P, and -

Syntax: **k**
Ctrl-P
(up arrow)

Function: Moves the cursor up one line, remaining in the same column. If a count is given, the cursor is moved *count* lines.

Syntax: **-**

Function: Moves the cursor up to the beginning of the previous line. If a count is given, the cursor is moved *count* lines.

Beginning of Line: 0 and ∧

Syntax: **∧**
0

Function: Moves the cursor to the beginning of the current line. Note that **0** always moves the cursor to the first character of the current line. The caret (∧) works somewhat differently: it moves to the first character on a line that is not a tab or a space. This is useful when editing files that have a great deal of indentation, such as program texts.

End of Line: -$

Syntax: **$**

Function: Moves the cursor to the end of the current line. Note that the cursor resides on top of the last character on the line. If a *count* is given, the cursor is moved forward *count*-1 lines to the end of the line.

Goto Line: -G

Syntax: [*linenumber*]**G**

Function: Moves the cursor to the beginning of the line specified by *linenumber*. If no *linenumber* is given, the cursor moves to the beginning of the *last* line in the file. To find the line number of the current line, use Ctrl-G.

Column: - |

Syntax: [*column*] **|**

Function: Moves the cursor to the column in the current line given by *column*. If no *column* is given, the cursor is moved to the first column in the current line.

Word Forward: -w and W

Syntax: **w**
W

Function: Mover the cursor right to the beginning of the next word. The lowercase **w** command searches for a word defined as a string of alphanumeric characters separated by punctuation or whitespace (that is, tab, newline, or space characters). The uppercase **W** command searches for a word defined as a string of non whitespace characters.

Back Word: -b and B

Syntax: **b**
B

Function: Moves the cursor left to the beginning of a word. The lowercase **b** command searches backward for a word defined as a string of alphanumeric characters separated by punctuation or whitespace (that is, tab, newline, or space characters). The uppercase **B** command searches for a word defined as a string of non whitespace characters. If the cursor is already within a word, it moves backward to the beginning of that word.

End: -e and E

Syntax: **e**
E

Function: Moves the cursor to the end of a word. The lowercase **e** command moves the cursor to the last character of a word, where a word is defined as a string of alphanumeric characters separated by punctuation or whitespace (that is, tab, newline, or space characters). The uppercase **E** moves the cursor to the last character of a word where a word is defined as a string of non-whitespace characters. If the cursor is already within a word, it moves to the end of that word.

Sentence: -(and)

Syntax: (
)

Function: Moves the cursor to the beginning (left parenthesis) or end of a sentence (right parenthesis). A sentence is defined as a sequence of characters ending with a period (.), question mark (?), or exclamation mark (!), followed by either two spaces or a newline. A sentence begins on the first non-whitespace character following a preceding sentence. Sentences are also delimited by paragraph and section delimiters. See below.

Paragraph: -{ and }

Syntax: }
{

Function: Moves the cursor to the beginning ({) or end (}) of a paragraph. A paragraph is defined with the *paragraphs* option. By default, paragraphs are delimited by the **nroff** macros ".IP," ".LP," ".P," ".QP," and ".bp". Paragraphs also begin after empty lines.

Section: -[[and]]

Syntax:]]
[[

Function: Moves the cursor to the beginning ([[) or end (]]) of a section. A section is defined with the *sections* option. By default, sections are delimited by the **nroff** macros ".NH" and ".SH". Sections also start at form feeds (Ctrl-L) and at lines beginning with a brace ({).

Match Delimiter: -%

Syntax: %

Function: Moves the cursor to a matching delimiter, where a delimiter is a parenthesis, a bracket, or a brace. This is useful when matching pairs of nested parentheses, brackets, and braces.

Home: -H

Syntax: [*offset*]**H**

Function: Moves the cursor to upper left corner of screen. Use this command to quickly move to the top of the screen. If an *offset* is given, the cursor is homed *offset* -1 number of lines from the top of the screen. Note that the command "dH" deletes all lines from the current line to the top line shown on the screen.

Middle Screen: -M

Syntax: **M**

Function: Moves the cursor to the beginning of the screen's middle line. Use this command to quickly move to the middle of the screen from either the top or the bottom. Note that the command "dM" deletes from the current line to the line specified by the **M** command.

Lower Screen: -L

Syntax: [*offset*]**L**

Function: Moves the cursor to the lowest line on the screen. Use this command to quickly move to the bottom of the screen. If an *offset* is given, the cursor is homed *offset* -1. number of lines from the bottom of the screen. Note that the command "dL" deletes all lines from the current line to the bottom line shown on the screen.

Previous Context: -"and"

Syntax: ''
'*character*
``
`*character*

Function: Moves the cursor to previous context or to context marked with the **m** command. If the single quotation mark or back quotation mark is doubled, the cursor is moved to previous context. If a single character is given after either quotation mark, the cursor is moved to the location of the specified mark as defined by the **m** command. Previous context is the location in the file of the last **non-relative** cursor movement. The single quotation mark (') syntax is used to move to the beginning of the line representing the previous context. The back quotation mark (') syntax is used to move to the previous context *within* a line.

The Screen Commands

The screen commands are not cursor movement commands and cannot be used in delete commands as the delimiters of text objects. However, the screen commands do move the cursor and are useful in paging or scrolling through a file. These commands are described below.

Page: - Ctrl-U and Ctrl-D

Syntax: [*size*]**Ctrl-U**
[*size*]**Ctrl-D**

Function: Scrolls the screen up a half window (Ctrl-U) or down a half window (Ctrl-D). If *size* is given, the scroll is *size* number of lines. This value is remembered for all later scrolling commands.

Scroll: - Ctrl-F and Ctrl-B

Syntax: **Ctrl-F**
Ctrl-B

Function: Pages screen forward and backward. Two lines of continuity are kept between pages if possible. A preceding count gives the number of pages to move forward or backward.

Status: - Ctrl-G

Syntax: **BELL**
Ctrl-G

Function: Prints **vi** status on status line. This gives you the name of the file you are editing, whether it has been modified, the current line number, the number of lines in the file, and the percentage of the file (in lines) that precedes the cursor.

Zero Screen: -z

Syntax: [*linenumber*]z[*size*]**Enter**
[*linenumber* | z[*size*].
[*linenumber*]z[*size*]-

Function: Redraws the display with the current line placed at or "zeroed" at the top, middle, or bottom of the screen, respectively. If you give a *size*, the number of lines displayed is equal to *size*. If a preceding *linenumber* is given, the given line is placed at the top of the screen. If the last argument is an Enter, the current line is placed at the top of the screen. If the last argument is a period (.), the current line is placed in the middle of the screen. If the last argument is a hyphen (-), the current line is placed at the bottom of the screen.

Redraw: - Ctrl-R or Ctrl-L

Syntax: **Ctrl-R**
Ctrl-L

Function: Redraws the screen. Use this command to erase any system messages that may disrupt your screen. Note that system messages do not affect the file you are editing.

Text Insertion

The text insertion commands always place you in insert mode. Exit from insert mode is always done by pressing Esc. The following insertion commands are "pure" insertion commands; no text is deleted when you use them. This differs from the text modification commands change, replace, and substitute, which delete and then insert text in one operation.

Insert: -i and I

Syntax: **i[***text***]ESC**
I[*text***]ESC**

Function: Insert *text* in editing buffer. The lowercase i command places you in insert mode. *Text* is inserted *before* the character beneath the cursor. To insert a newline, just press an Enter. Exit insert mode by typing the Esc key. The uppercase **I** command places you in insert mode, but begins text insertion at the beginning of the current line, rather than before the cursor.

Append: -a and A

Syntax: **a[***text***]ESC**
A[*text***]ESC**

Function: Appends *text* to the editing buffer. The lowercase **a** command works *exactly* like the lowercase **i** command, except that text insertion begins after the cursor and not before. This is the one way to add text to the end of a line. The uppercase **A** command begins appending text at the end of the current line rather than after the cursor.

Open New Line: -o and O

Syntax: **o[***text***]ESC**
O[*text***]ESC**

Function: Opens a new line and inserts text. The lowercase **o** command opens a new line below the current line; uppercase **O** opens a new line *above* the current line. After the new line has been opened, both these commands work like the **I** command.

Text Deletion

Many of the text deletion commands use the letter **d** as an operator. This operator deletes text objects delimited by the cursor and a cursor movement command. Deleted text is always saved away in a buffer. The delete commands are described below:

Delete Character: -x and X

Syntax: **x**
X

Function: Deletes a character. The lowercase **x** command deletes the character beneath the cursor. With a preceding count, *count* characters are deleted to the right beginning with the character beneath the cursor. This is a quick and easy way to delete a few characters. The uppercase **X** command deletes the character just before the cursor. With a preceding count, *count* characters are deleted backward, beginning with the character just before the cursor.

Delete: -d and D

Syntax: **d***cursor-movement*
dd
D

Function: Deletes a text object. The lowercase **d** command takes a *cursor-movement* as an argument. If the *cursor-movement* is an intraline command, deletion takes place from the cursor to the end of the text object delimited by the *cursor-movement*. Deletion forward deletes the character beneath the cursor; deletion backward does not. If the *cursor-movement* is a multiline command, deletion takes place from and including the current line to the text object delimited by the *cursor-movement*. The **dd** command deletes whole lines. The uppercase **D** command deletes from and including the cursor to the end of the current line.

Deleted text is automatically pushed on a stack of buffers numbered 1 through 9. The most recently deleted text is also placed in a special delete buffer that is logically buffer Ø. This special buffer is the default buffer for all (put) commands using the double quotation mark (") to specify the number of the buffer for delete, put, and yank commands. The buffers 1 through 9 can be accessed with the **p** and **P** (put) commands by appending the double quotation mark (") to the number of the buffer. For example:

```
"4p
```

puts the contents of delete buffer number 4 in your editing buffer just below the current line. Note that the last deleted text is "put" by default and does not need a preceding buffer number.

Text Modification

The text modification commands all involve the replacement of text with other text. This means that some text will necessarily be deleted. All text modification commands can be "undone" with the **u** command, discussed below:

Undo: -u and U

Syntax: **u**
U

Function: Undoes the last insert or delete command. The lowercase **u** command undoes the last insert or delete command. This means that after an insert, **u** deletes text; and after a delete, **u** inserts text. For the purposes of undo, all text modification commands are considered insertions.

The uppercase **U** command restores the current line to its state before it was edited, no matter how many times the current line has been edited since you moved to it.

Repeat: - .

Syntax:	**.**
Function:	Repeats the last insert or delete command. A special case exists for repeating the **p** and **P** "put" commands. When these commands are preceded by the name of a delete buffer, successive **u** commands print out the contents of the delete buffers.

Change:: -c and C

Syntax:	**c***cursor-movement text***Esc** **C***text***Esc** **cc***text***Esc**
Function:	Changes a text object and replaces it with *text* . Text is inserted as with the **i** command. A dollar sign ($) marks the extent of the change. The **c** command changes arbitrary text objects delimited by the cursor and a *cursor-movement.* The **C** and **cc** commands affect whole lines and are identical in function.

Replace: -r and R

Syntax:	**r***char* **R***text***ESC**
Function:	Overstrikes character or line with *char* or *text,* respectively. Use **r** to overstrike a single character and **R** to overstrike a whole line. A *count* multiplies the replacement text *count* times.

Substitute: -s and S

Syntax: s*text*Esc
S*text*Esc

Function: Substitutes current character or current line with *text*. Use s to replace a single character with new text. Use S to replace the current line with new text. If a preceding count is given, *text* substitutes for count number of characters or lines depending on whether the command is s or S, respectively.

Filter: -!

Syntax: *!cursor-movement cmdEnter*

Function: Filters the text object delimited by the cursor and *cursor-movement* through the XENIX command, *cmd*. For example, the following command sorts all lines between the cursor and the bottom of the screen, substituting the designated lines with the sorted lines:

```
!Lsort
```

Arguments and shell metacharacters may be included as part of *cmd*; however, standard input and output are always associated with the text object being filtered.

Join Lines: -J

Syntax: **J**

Function: Joins the current line with the following line. If a count is given, then count lines are joined.

Shift: -< and >

Syntax:
>[*cursor-movement*]
<[*cursor-movement*]
>>
<<

Function: Shifts text left (>) or right (<). Text is shifted by the value of the option *shiftwidth*, which is normally set to eight spaces. Both the > and < commands shift all lines in the text object delimited by the current line and *cursor-movement*. The >> and << commands affect whole lines. All versions of the command can take a preceding *count* that acts to multiply the number of objects affected.

Text Movement

The text movement commands move text in and out of the named buffers a-z and out of the delete buffers 1-9. These commands either "yank" text out of the editing buffer and into a named buffer or "put" text into the editing buffer from a named buffer or a delete buffer. By default, text is put and yanked from the "unnamed buffer," which is also where the most recently deleted text is placed. Thus it is quite reasonable to delete text, move your cursor to the location where you want the deleted text placed and put the text back into the editing buffer at this new location with the **p** or **P** command.

The named buffers are most useful for keeping track of several pieces of text that you want to keep on hand for later access, movement, or rearrangement. These buffers are named with the letters "a" through "z." To refer to one of these buffers (or one of the numbered delete buffers) in a command such as put, yank, or delete, use a quotation mark. For example, to yank a line into the buffer named *a*, type:

```
"ayy
```

To put this text back into the file, type:

```
"ap
```

If you yank text into the buffer named *A* rather than *a*, text is appended to the buffer.

The contents of the named buffers are not destroyed when you switch files. Therefore, you can delete or yank text into a buffer, switch files, and then do a put. Buffer contents are *destroyed* when you exit the editor.

Put: -p and P

Syntax:	[*"alphanumeric*]**p** [*"alphanumeric*]**P**
Function:	Puts text from a buffer into the editing buffer. If no buffer name is specified, text is put from the unnamed buffer. The lowercase **p** command puts text either below the current line or after the cursor, depending on whether the buffer contains a partial line or not. The uppercase **P** command puts text either above the current line or before the cursor, again depending on whether the buffer contains a partial line or not.

Yank: -y and Y

Syntax: [*"letter*]y*cursor-movement*
[*"letter*]**yy**
[*"letter*]**Y**

Function: Copies text in the editing buffer to a named buffer. If no buffer name is specified, text is yanked into the unnamed buffer. If an uppercase *letter* is used, text is appended to the buffer and does not overwrite and destroy the previous contents. When a *cursor-movement* is given as an argument, the delimited text object is yanked. The **Y** and yy commands yank a single line. If a preceding count is given, multiple lines are yanked.

Searching

The search commands search forward or backward in the editing buffer for text that matches a given regular expression.

Search: -/ and ?

Syntax: /[*pattern*]/[*offset*]**Enter**
/[*pattern*]**Enter**
?[*pattern*]?[*offset*]**Enter**
? [*pattern*]**Enter**

Function: Searches forward (/) or backward (?) for *pattern.* A string is actually a regular expression. The trailing delimiter is not required. If no *pattern* is given, the last *pattern* searched for is used. After the second delimiter, an *offset* may be given, specifying the beginning of a line relative to the line on which *pattern* was found.

For example:

```
/word/-
```

finds the beginning of the line immediately preceding the line containing "word" and

```
/word/+2
```

finds the beginning of the line two lines after the line containing "word." See also the *ignorecase* and *magic* options.

Next String: -n and N

Syntax: **n**
N

Function: Repeats the last search command. The **n** command repeats the search in the same direction as the last search command. The **N** command repeats the search in the opposite direction of the last search command.

Find Character: -f and F

Syntax: **f***char*
F*char*
;
,

Function: Finds character *char* on the current line. The lowercase **f** searches forward on the line; the uppercase **F** searches backward. The semicolon (**;**) repeats the last character search. The comma (**,**) reverses the direction of the search.

To Character: -t and T

Syntax: t*char*
T*char*
;
,

Function: Moves the cursor up to but not on to *char*. The semicolon (;) repeats the last character search. The comma (,) reverses the direction of the search.

Mark: -m

Syntax: m*letter*

Function: Marks a place in the file with a lowercase *letter*. You can move to a mark using the "to mark" commands described below. It is often useful to create a mark, move the cursor, and then delete from the cursor to the mark *a* with the following command:

```
d'a
```

To Mark: - ´and`

Syntax: *´letter*
`letter

Function: Move to *letter*. These commands let you move to the location of a mark. Marks are denoted by single lowercase alphabetic characters. Before you can move to a mark, it must first be created with the **m** command. The back quotation mark (´) moves you to the exact location of the mark within a line; the forward quotation mark (`) moves you to the beginning of the line containing the mark. Note that these commands are also legal cursor movement commands.

Exit and Escape Commands

There are several commands that are used to escape from vi command mode and to exit the editor. These are described below: **ex Escape: -:**

Syntax: **:**

Function: Enters **ex** escape mode to execute an **ex** command. The colon appears on the status line as a prompt for an **ex** command. You then can enter an **ex** command line terminated by either an Enter or an Esc and the **ex** command will execute. You are then prompted to type Enter to return to **vi** command mode. During the input of the **ex** command line or during execution of the **ex** command you may press Interrupt (Del) to stop what you are doing and return to **vi** command mode.

Exit Editor: - ZZ

Syntax: **ZZ**

Function: Exit **vi** and write out the file if any changes have been made. This returns you to the shell from which you invoked **vi.**

Quit to Ex: -Q

Syntax: **Q**

Function: Enters the **ex** editor. When you do this, you are still editing the same file. You can return to **vi** by typing the **vi** command from **ex**.

ex Commands

Typing the colon (:) escape command when in command mode, produces a colon prompt on the status line. This prompt is for a command available in the line-oriented editor, **ex**. In general, **ex** commands let you write out or read in files, escape to the shell, or switch editing files.

Many of these commands perform actions that affect the "current" file by default. The current file is normally the file that you named when you invoked **vi**, although the current file can be changed with the "file" command, **f**, or with the "next" command, **n**. In most respects, these commands are identical to similar commands for the editor, **ed**. All such **ex** commands are ended by either an Enter or an Esc. We shall use an Enter in our examples. Command entry is terminated by typing an Interrupt (Del).

Command Structure

Most **ex** command names are English words, and initial prefixes of the words are acceptable abbreviations. In descriptions, only the abbreviation is discussed, because this is the most frequently used form of the command. The ambiguity of abbreviations is resolved in favor of the more commonly used commands. As an example, the command **substitute** can be abbreviated **s** while the shortest available abbreviation for the **set** command is **se:**.

Most commands accept prefix addresses specifying the lines in the file that they are to affect. A number of commands also may take a trailing *count* specifying the number of lines to be involved in the command. Counts are rounded down if necessary. Thus, the command **5p** prints the fifth line in the buffer while **move 5** moves the current line after line 5.

Some commands take other information or parameters, stated after the command name. Examples might be option names in a **set** command, such as **set** *number*, a *filename* in an **edit** command, a regular expression in a **substitute** command, or a target address for a **copy** command, such as:

```
1,5 copy 25
```

A number of commands have variants. The variant form of the command is invoked by placing an exclamation mark (!) immediately after the command name. Some of the default variants may be controlled by options; in this case, the exclamation mark turns off the meaning of the default.

In addition, many commands take flags, including the characters **p** and **l**. A **p** or **l** must be preceded by a blank or tab. In this case, the command abbreviated by these characters is executed after the command completes. Since **ex** normally prints the new current line after each change, **p** is rarely necessary. Any number of plus (+) or minus (-) characters may also be given with these flags. If they appear, the specified offset is applied to the current line value before the printing command is executed.

Most commands that change the contents of the editing buffer give feedback if the scope of the change exceeds a threshold given by the **report** option. This feedback helps to detect undesirably large changes so that they may be quickly and easily reversed with the **undo** command. After commands with global effect, you will be informed if the net change in the number of lines in the buffer during this command exceeds this threshold.

Command Addressing

The following specify the line addressing syntax for **ex** commands:

.	The current line. Most commands leave as the current line the last line they affect. The default address for most commands is the current line, thus . is rarely used alone as an address.

n	The *n*th line in the editing buffer, lines being numbered sequentially from 1.
$	The last line in the buffer.
%	An abbreviation for "1,$," the entire buffer.
+*n* **or** -*n*	An offset, *n* relative to the current buffer line. The forms ".+3" "+3" and "+++" are all equivalent. If the current line is line 100, they all address line 103.

/pattern/ **or** *?pattern?*

Scan forward and backward respectively for text matching the regular expression given by *pattern.* Scans normally wrap around the end of the buffer. If all that is desired is to print the next line containing *pattern,* the trailing slash (/) or question mark (?) may be omitted. If *pattern* is omitted or explicitly empty, the string matching the last specified regular expression is located. The forms **Enter** and **?Enter** scan using the last named regular expression. After a substitute, **Enter** and **??Enter** would scan using that substitute's regular expression.

+″**or** ′ *x*

Before each non-relative motion of the current line dot (.), the previous current line is marked with a label, subsequently referred to with two single quotation marks (″). This makes it easy to refer or return to this previous context. Marks are established with the **vi m** command, using a single lowercase letter as the name of the mark. Marked lines are later referred to with the notation

x.

where *x* is the name of a mark.

Addresses to commands consist of a series of addresses, separated by a colon (,) or a semicolon (;). Such address lists are evaluated left to right. When addresses are separated by a semicolon (;), the current line (.) is set to the value of the previous addressing expression before the next address is interpreted. If more addresses are given than the command requires, all but the last one or two are ignored. If the command takes two addresses, the first addressed line must precede the second in the buffer. Null address specifications are permitted in a list of addresses, the default in this case is the current line "."; thus "**,1ØØ**" is equivalent to "**.,1ØØ**". It is an error to give a prefix address to a command which expects none.

Command Format

The following is the format for **ex** commands:

[*address*] [*command*] [*!*][*parameters*][*count*] [*flags*]

All parts are optional depending on the particular command and its options. The following section describes specific commands.

Argument List Commands

The argument list commands allow you to work on a set of files by remembering the list of filenames that are specified when you invoke **vi**. The **args** command lets you examine this list of filenames. The **file** command gives you information about the current file. The **n** (next) command lets you either edit the next file in the argument list or change the list. And the **rewind** command lets you restart editing the files in the list. All of these commands are described below:

args The members of the argument list are printed, with the current argument delimited by brackets. For example, a list might look like this:

```
file1 file2 [file3] file4 file5
```

The current file is file3

f	Prints the current filename, whether it has been modified since the last **write** command, whether it is read-only, the current line number, the number of lines in the buffer, and the percentage of the buffer that you have edited. In the rare case that the current file is "[Not edited]" this is noted also; in this case you have to use the form "**w!**" to write to the file, because the editor is not sure that a **w** command will not destroy a file unrelated to the current contents of the buffer.
f *file*	The current filename is changed to *file* which is considered "[Not edited]".
n	The next file in the command line argument list is edited.
n!	This variant suppresses warnings about the modifications to the buffer not having been written out, discarding irretrievably any changes that may have been made.
n [+*command*]*filelist*	The specified *filelist* is expanded and the resulting list replaces the current argument list; the first file in the new list is then edited. If *command* is given (it must contain no spaces), it is executed after the first file is edited.
rew	The argument list is rewound, and the first file in the list is edited.
rew!	The argument list is rewound and any changes made to the current buffer are discarded.

Edit Commands

To edit a file other than the one you are currently editing, you will often use one of the variations of the **e** command.

In the following discussions, note that the name of the current file is always remembered by **vi** and is specified by a percent sign (%). The name of the *previous* file in the editing buffer is specified by a number sign (#).

The edit commands are described below:

e *file*	Used to begin an editing session on a new file. The editor first checks to see if the buffer has been modified since the last **w** command was issued. If it has been, a warning is issued and the command is not executed. The command otherwise deletes the entire contents of the editing buffer, makes the named file the current file, and prints the new filename. After ensuring that this file is sensible, (that is, that it is not a binary file, directory, or a device), the editor reads the file into its buffer. If the read of the file finishes without error, the number of lines and characters read is printed on the status line. If there were any non-ASCII characters in the file, they are stripped of their non-ASCII high bits, and any null characters in the file are discarded. If none of these errors occurred, the file is considered edited. If the last line of the input file is missing the trailing newline character, it is supplied and an error message issued. The current line is initially the first line of the file.
e! *file*	This variant suppresses the complaint about modifications having been made and not written from the editor buffer, thus discarding all changes that have been made before editing the new file.

e *+n file* Causes the editor to begin editing at line *n* rather than at the first line. The argument *n* may also be an editor command containing no spaces; for example, "+/pattern".

Ctrl-^ This is a shorthand equivalent for an **:e** #Enter, which returns to the previous position in the last edited file. If you do not want to write the file you should use **:e!** #Enter. instead.

Write Commands

The write commands let you write out all or part of your editing buffer to either the current file or to some other file. These commands are described below:

w *file*

Writes changes made back to *file*, printing the number of lines and characters written. Normally, *file* is omitted and the buffer is written to the name of the current file. If *file* is specified, text is written to that file. The editor writes to a file only if it is the current file and is edited or if the file does not exist. Otherwise, you must give the variant form **w!** to force the write. If the file does not exist it is created. The current filename is changed only if there is no current filename; the current line is never changed.

If an error occurs while the current and *edited* file is being written, the editor prints:

```
No write since last change
```

even if the buffer had not previously been modified.

w *>>file*

Appends the buffer contents at the end of an existing file. Previous file contents are not destroyed.

w! *name*

Overrides the checking of the normal **write** command, and writes to any file that the system permits.

w! *command*

Writes the specified lines into *command.* Note the difference between:

w!*file*

which overrides checks and

w!*cmd*

which writes to a command. The output of this command is displayed on the screen and not inserted in the editing buffer.

Read Commands

The read commands let you read text into your editing buffer at any location you specify. The text you read in must be at least one line long, and can be either a file or the output from a command.

r *file*	Places a copy of the text of the given file in the editing buffer after the specified line. If no file is given, the current filename is used. The current filename is not changed unless there is none, in which case the file becomes the current name. If the file buffer is empty and there is no current name, this is treated as an **e** command. Address 0 is legal for this command and causes the file to be read at the beginning of the buffer. Statistics are given as for the **e** command when the **r** successfully terminates. After an **r**, the current line is the last line read.
r !*command*	Reads the output of *command* into the buffer after the specified line. A blank or tab before the exclamation mark (!) is mandatory.

Quit Commands

There are several ways to exit **vi**. Some end the editing session, some write out the editing buffer before exiting, and some warn you if you decide to exit without writing out the buffer. All of these ways of exiting are described below:

q	Exits **vi**. No automatic write of the editing buffer to a file is performed. However, **vi** issues a warning message if the file has changed since the last **w** command was issued, and does not quit. Also, **vi** issues a diagnostic message if there are more files in the argument list left to edit. Normally, you want to save your changes, and you should give a **w** command. If you want to discard them, use the "q!" command variant.
q!	Quits from the editor, discarding changes to the buffer without a message.
wq *name*	Like a **w** and then a **q** command.
wq! *name*	This variant overrides checking of the **w** command so that you can write to a file that the system allows.
x *name*	If any changes have been made and not written, writes the buffer out and quits. Otherwise, it just quits.

The global and substitute commands allow you to perform complex changes to a file in a single command. Learning how to use these commands is a must for the serious user of **vi**.

g/*pattern*/*cmds*	The **g** command has two distinct phases. In the first phase, each line matching *pattern* in the editing buffer is marked. Next, the given command list is executed with the current line, dot (.), initially set to each marked line. The command list consists of the remaining commands on the current input line and may continue to multiple lines by ending all but the last such line with a backslash (\\). This multiple-line option will not work from within **vi**, you must switch to **ex** to do it. If *cmds* (or the trailing slash (/) delimiter) is omitted, each line matching *pattern* is printed. The **g** command itself may not appear in *cmds*. The options *autoprint* and *autoindent* are inhibited during a global command and the value of the *report* option is temporarily infinite, in deference to a *report* for the entire global command. Finally, the context mark (\`) or (′) is set to the value of the current line (.) before the global command begins and is not changed during a global command.

The following global commands, most of them substitutions, cover the most frequent uses of the global command.

g/*s1*/**p**	This command simply prints all lines that contain the string *s1*.
g/*s1*/**s**//*s2*/	This command substitutes the *first* occurrence of *s1* on all lines that contain it with the string *s2*.

g/*s1***/s//***s2***/g**	This command substitutes all occurrences of *s1* with the string *s2*. This includes multiple occurrences of *s1* on a line.
g/*s1***/s//***s2***/gp**	This command works the same as the preceding example, except that in addition, all changed lines are printed on the screen.
g/*s1***/s//***s2***/gc**	This command asks you to confirm that you want to make each substitution of the string *s1* with the string *s2*. If you type a *y* the given substitution is made, otherwise it is not.
g/*s0***/s/***s1***/***s2***/g**	This command marks all lines that contain the string *s0*, and then, for those lines only, it substitutes all occurrences of the string *s1* with *s2*.
g!/*pattern*/*cmds*	This variant form of *g* runs *cmds* at each line not matching *pattern*

s/*pattern*/*repl*/*options*

On each specified line, the first instance of text matching the regular expression *pattern* is replaced by the replacement text *repl*. If the *global* indicator option character **g** appears, all instances on a line are substituted. If the *confirm* indication character **c** appears, before each substitution, the line to be substituted is printed on the screen with the string to be substituted marked with caret (^) characters. By typing a **y**, you cause the substitution to be performed; any other input causes no change to take place. After an **s** command the current line is the last line substituted.

v/*pattern*/*cmds*	A synonym for the global command variant **g!**, running the specified *cmds* on each line that does not match *pattern*.

Text Movement Commands

The text movement commands are largely superseded by commands available in **vi** command mode. However, the following two commands are still quite useful.

co *addr flags*	A copy of the specified lines is placed after *addr*, which may be "Ø". The current line "." addresses the last line of the copy.
[*range*]**m***addr*	The **m** command moves the lines specified by *range* after the line given by *addr*. For example, "m+" swaps the current line and the following line, because the default range is just the current line. The first of the moved lines becomes the current line (.).

Shell Escape Commands

You will often want to escape from the editor to execute normal XENIX commands. You may also want to change your working directory so that your editing can be done with respect to a different working directory. These operations are described below:

cd *directory*	The specified *directory* becomes the current directory. If no directory is specified, the current value of the *home* option is used as the target directory. After a **cd** the current file is not considered to have been edited so that write restrictions on preexisting files still apply.
sh	A new shell is created. You may invoke as many commands as you like in this shell. To return to **vi**, type a Ctrl-D to terminate the shell.

!command The remainder of the line after the exclamation (!) is sent to a shell to be executed. Within the text of *command*, the characters "%" and "#" are expanded as the filenames of the current file and the last edited file and the character "!" is replaced with the text of the previous command. Thus, in particular, "!!" repeats the last such shell escape. If any such expansion is performed, the expanded line is echoed. The current line is unchanged by this command.

If there has been "[No write]" of the buffer contents since the last change to the editing buffer, a diagnostic is printed as a warning before the command is executed. A single exclamation (!) is printed when the command finishes.

Other Commands

The following command descriptions explain how to use miscellaneous **ex** commands that do not fit into the above categories:

abbr Maps the first argument to the following string. For example, the following command:

```
:abbr rainbow yellow green blue red
```

maps "rainbow" to "yellow green blue red". Abbreviations can be turned off with the **unabbreviate** command, as in:

```
:una rainbow
```

map, map!
Maps any character or escape sequence to an existing command sequence. Characters mapped with **map!** work only in insert mode, while characters mapped with **map** work only in command mode.

nu Prints each specified line preceded by its buffer line number. The current line is left at the last line printed. To get automatic line numbering of lines in the buffer, set the *number* option.

preserve The current editing buffer is saved as though the system had just gone down. This command is for use only in emergencies when a **w** command has resulted in an error and you don't know how to save your work.

= Prints the line number of the addressed line. The current line is unchanged.

recover *file*

Recovers *file* from the system save area. The system saves a copy of the editing buffer only if you have made changes to the file, thc system goes down, or you execute a **preserve** command. Except when you use **preserve**, you will be notified by mail when a file is saved.

set *argument*

With no arguments, **set** prints these options whose values have been changed from their defaults; with the argument *all* it prints all of the option values.

Giving an option name followed by a question mark (**?**) causes the current value of that option to be printed. The **?** is unnecessary unless the option is Boolean valued. Switch options are given values either with:

set *option*

to turn them on or:

set *nooption*

to turn them off. String and numeric options are assigned with:

set *option=value*

More than one parameter may be given to **set**; all are interpreted from left to right.

tag *label* The focus of editing switches to the location of *label*. If necessary, **vi** will switch to a different file in the current directory to find *label*. If you have modified the current file before giving a **tag** command, you must first write it out. If you give another **tag** command with no argument, the previous *label* is used.

Similarly, if you type only a Ctrl-], **vi** searches for the word immediately after the cursor as a tag. This is equivalent to typing ":tag", the word, and then an Enter.

The tags file is normally created by a program such as **ctags** and consists of a number of lines with three fields separated by blanks or tabs. The first field gives the name of the tag, the second the name of the file where the tag resides, and the third an addressing form that can be used by the editor to find the tag. This field is usually a contextual scan using */pattern/* to be immune to minor changes in the file. Such scans are always performed as if the *nomagic* option was set. The tag names in the tags file must be sorted alphabetically. There are a number of options that can be set to affect the **vi** environment. These can be set with the ex **set** command either while editing or immediately after **vi** is invoked in the **vi** start-up file, *.exrc*.

The first thing that must be done before you can use **vi**, is to set the terminal type so that **vi** understands how to talk to the particular terminal you are using.

Each time **vi** is invoked, it reads commands from the file named *.exrc* in your home directory. This file normally sets the user's preferred options so that they need not be set manually each time you invoke **vi**. Each of the options is described in detail below.

Options

There are only two kinds of options: switch options and string options. A switch option is either on or off. A switch is turned off by prefixing the word **no** to the name of the switch within a **set** command. String options are strings of characters that are assigned values with the syntax *option=string*. Multiple options may be specified on a line. Options are listed below:

autoindent, ai default: **noai**

Can be used to ease the preparation of structured program text. For each line created by an append, change, insert, open, or substitute operation, **vi** looks at the preceding line to determine and insert an appropriate amount of indentation. To back the cursor up to the preceding tab stop, you can type Ctrl-D. The tab stops going backward are defined as multiples of the **shiftwidth** option. You cannot backspace over the indent, except by typing a Ctrl-D.

Specially processed in this mode is a line with no characters added to it, which turns into a completely blank line (the whitespace provided for the **autoindent** is discarded.) Also specially processed in this mode are lines beginning with a caret (^) and immediately followed by a Ctrl-D. This causes the input to be repositioned at the beginning of the line but retains the previous indent for the next line. Similarly, a Øfollowed by a Ctrl-D repositions the cursor at the beginning but without retaining the previous indent. **Autoindent** doesn't happen in global commands.

autoprint, ap default: **ap**

Causes the current line to be printed after each **ex** copy, move, or substitute command. This has the same effect as supplying a trailing **p** to each such command. The **autoprint** option is suppressed in globals and only applies to the last of many commands on a line.

autowrite, aw default: **noaw**

Causes the contents of the buffer to be automatically written to the current file if you have modified it when you give a **next**, **rewind**, **tag**, or **!** command, or a Ctrl-^ (switch files) or Ctrl-] (tag go to) command.

beautify, bf default: **nobeautify**

Causes all control characters except tab, new line and form feed to be discarded from the input. A message is returned the first time a backspace character is discarded. The **beautify** option does not apply to command input.

directory, dir default: **dir=/tmp**

Specifies the directory in which **vi** places the editing buffer file. If this directory is not writable, the editor exits abruptly when it fails to write to the buffer file.

edcompatible default:**noedcompatible**

Causes the presence or absence of **g** and **c** suffixes on substitute commands to be remembered and to be toggled on and off by repeating the suffixes. The suffix **r** causes the substitution to be like the (~) command, instead of like **&**.

errorbells,eb default: **noeb**

Error messages usually are preceded by a beep. If possible, the editor places the error message in inverse video instead of sounding the beep.

hardtabs, ht default: **ht=8**

Gives the boundaries on which terminal tabs are set or on which the system expands tabs.

ignorecase, ic default:**noic**

Maps all uppercase characters in the text to lowercase in regular expression matching. In addition, all uppercase characters in regular expressions are mapped to lowercase except in character class specifications enclosed in brackets.

lisp default: **nolisp**

Autoindent indents appropriately for LISP code, and the **( ) {} [[** and **]]** commands are modified to have meaning for LISP.

list default: **nolist**

All printed lines are displayed unambiguously, showing tabs and end-of-lines.

magic default: **magic**

If **nomagic** is set, the number of regular expression special characters is greatly reduced, with only up-arrow (↑), and dollar sign ($) having special effects. In addition, the special characters "~" and "&" in replacement patterns are treated as normal characters. All the normal special characters may be made **magic** when **nomagic** is set by preceding them with a backslash (\).

mesg default: **nomesg**

Causes write permission to be turned off to the terminal while you are in visual mode, if **nomesg** is set. This prevents people writing to your screen with the XENIX **write** command and disrupting your screen as you edit.

number, n default: **nonumber**

Causes all output lines to be printed with their line numbers.

open default: **open**

If set to **noopen**, the commands **open** and **visual** are not permitted from ex. This is set to prevent confusion resulting from accidental entry to open or visual mode.

optimize, opt default: **optimize**

Output of text to the screen is expedited by setting the terminal so that it does not perform automatic carriage returns when printing more than one line of output, thus greatly speeding output on terminals without addressable cursors when text with leading whitespace is printed.

paragraphs, para default: **para=IPLPPPQPP TPbp**

Specifies paragraph delimiters for the { and } operations. The pairs of characters in the option's value are the names of the **nroff** macros that start paragraphs.

prompt default: **prompt**

Input to **ex** is prompted for with a colon (:). If **noprompt** is set, when **ex** command mode is entered with the **Q** command, no colon prompt is displayed on the status line.

redraw default: **noredraw**

The editor simulates (using great amounts of output), an intelligent terminal on a dumb terminal. Useful only at very high speed.

remap default: **remap**

If on, mapped characters are repeatedly tried until they are unchanged. For example, if *o* is mapped to *O* and *O* is mapped to *I*, *o* will map to *I* if remap is set, and to *O* if **noremap** is set.

report default: **report=5**

Specifies a threshold for feedback from commands. Any command that modifies more than the specified number of lines will provide feedback about the scope of its changes. For global commands and the **undo** command, which have potentially far reaching scope, the net change in the number of lines in the buffer is presented at the end of the command, subject to this same threshold. Thus, notification is suppressed during a **g** command on the individual commands performed.

scroll default: **scroll= window**

Determines the number of logical lines scrolled when Ctrl-D is received from a terminal input in command mode, and the number of lines printed by a command mode **z** command (double the value of *scroll*).

sections default: **sections=SHNHH HU**

Specifies the section macros for the **[[** and **]]** operations. The pairs of characters in the option's value are the names of the **nroff** macros that start paragraphs.

shell, sh default: **sh=/bin/sh**

Gives the path name of the shell forked for the shell escape command "!", and by the **shell** command. The default is taken from SHELL in the environment, if present.

shiftwidth, sw default: **sw=8**

Gives the width of a software tab stop, used in reverse tabbing with Ctrl-D when using **autoindent** to append text, and by the shift commands.

showmatch, sm default: **nosm**

When a) or } is typed, moves the cursor to the matching (or { for one second if this matching character is on the screen.

tabstop, ts default: **ts=8**

The editor expands tabs in the input file to be on **tabstop** boundaries for the purposes of display.

taglength, tl default: **tl=0**

The first **taglength** characters in a tag name are significant, but all others are ignored. A value of 0 (the default) means that all characters are significant.

tags default: **tags=tags /usr/lib/tags**

A path of files to be used as tag files for the **tag** command. A requested tag is searched for in the specified files, sequentially. By default, files named *tag* are searched for in the current directory and in /usr/lib.

term default:= **value of shell TERM variable**
The terminal type of output device.

terse default: **noterse**
Shorter error diagnostics are produced for the experienced user.

warn default: **warn**
Warn if there has been "[No write since last change]" before a shell escape command (!).

window default: **window=** *speed dependent*
This specifies the number of lines in a text window. The default is 8 at slow speeds (600 baud or less), 16 at medium speed (1200 baud), and the full screen (minus one line) at higher speeds.

w300, w1200, w9600
These are not true options but set **window** (above) only if the speed is slow (300), medium (1200), or high (9600), respectively.

wrapscan, ws default: **ws**
Searches using the regular expressions in addressing wrap around past the end of the file.

wrapmargin, wm default: **wm=0**
Defines the margin for automatic insertion of new lines during text input. A value of Ø specifies no wrap margin.

writeany, wa default: **nowa**
Inhibits the checks normally made before **write** commands, allowing a write to any file that the system protection mechanism will allow.

Regular Expressions

A regular expression specifies a set of strings of characters. A member of this set of strings is said to be **matched** by the regular expression. The **vi** editor remembers two previous regular expressions: the previous regular expression used in a substitute command and the previous regular expression used elsewhere, referred to as the previous *scanning* regular expression. The previous regular expression can always be referred to by a null regular expression: for example, // or ??.

The regular expressions allowed by **vi** are constructed in one of two ways, depending on the setting of the **magic** option. The **ex** and **vi** default setting of **magic** gives quick access to a powerful set of regular expression special characters. The disadvantage of **magic** is that the user must remember that these special characters are **magic** and precede them with the backslash (\) to use them as "ordinary" characters. With **nomagic** set, regular expressions are much simpler, there being only three special characters. The power of the other special characters is still available by preceding the now ordinary character with a \. Note that \ is thus always a special character. In this discussion the magic **option** is assumed. With **nomagic** the only special characters are the caret (^) at the beginning of a regular expression, the dollar sign ($) at the end of a regular expression, and the backslash (\). The tilde (~) and the ampersand (&) also lose their special meanings related to the replacement pattern of a substitute.

The following basic constructs are used to construct **magic** mode regular expressions.

char An ordinary character matches itself. Ordinary characters are any characters except a caret (^) at the beginning of a line, a dollar sign ($) at the end of line, a asterisk (*) as any character other than the first, and any of the following characters:

. \ [ ] ~

These characters must be escaped (that is, preceded) by a backslash (\) if they are to be treated as ordinary characters.

^ At the beginning of a pattern, this forces the match to succeed only at the beginning of a line.

$ At the end of a regular expression this forces the match to succeed only at the end of the line.

. Matches any single character except the newline character.

\< Forces the match to occur only at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> Similar to \<, but matching the end of a "word," that is either the end of the line or before a character which is not a letter, a digit, or the underline character.

[*string*]
Matches any single character in the class defined by *string*. Most characters in *string* define themselves. A pair of characters separated by a hyphen (-) in *string* defines the set of characters between the specified lower and upper bounds, thus "[a-z]" as a regular expression matches any single lowercase letter. If the first character of *string* is a caret (^), the construct matches those characters which it otherwise would not. Thus [^a-z] matches anything but a lowercase letter or a newline. To place any of the characters caret, left bracket, or dash in *string* they must be escaped with a preceding backslash (\).

The concatenation of two regular expressions first matches the leftmost regular expression and then the longest string that can be recognized as a regular expression. The first part of this new regular expression matches the first regular expression and the second part matches the second. Any of the single character matching regular expressions mentioned above may be followed by an asterisk (*) to form a regular expression that matches zero or more adjacent occurrences of the characters matched by the prefixing regular expression. The tilde (~) may be used in a regular expression to match the text that defined the replacement part of the last s command. A regular expression may be enclosed between the sequences \(and \) to remember the text matched by the enclosed regular expression. This text can later be interpolated into the replacement text using the notation:

\ *digit*

where *digit* enumerates the set of remembered regular expressions.

The basic special characters for the replacement pattern are the ampersand (&) and the tilde (~). These are given as \& and \~ when **nomagic** is set. Each instance of the ampersand is replaced by the characters matched by the regular expression. In the replacement pattern, the tilde stands for the text of the previous replacement pattern.

Other metasequences possible in the replacement pattern are always introduced by a backslash (\). The sequence \ *n* is replaced by the text matched by the *n*th regular subexpression enclosed between \(and \). When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. The sequences \u and \l cause the immediately following character in the replacement to be converted to uppercase or lowercase, respectively, if this character is a letter. The sequences \U and \L turn such conversion on, either until \E or "\e" is encountered, or until the end of the replacement pattern.

Limitations

When using **vi**, you should note the following limits:

250,000 lines in a file

1024 characters per line

256 characters per global command list

128 characters per filename

128 characters in the previously inserted and deleted text

100 characters in a shell escape command

63 characters in a string valued option

30 characters in a tag name

Credit

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

Comments

The **/usr/lib/ex2.13preserve** program can be used to restore **vi** buffer files that were lost as a result of the system going down. The program searches the **/tmp** directory for **vi** buffer files and places them in the directory **/usr/preserve**. The owner can retrieve these files using the **-r** option.

The **/usr/lib/ex2.13preserve** program must be placed in the system startup file, **/etc/rc,** before the command that cleans out the **/tmp** directory. See the *XENIX System Administration* for more information on **/etc/rc**.

VSH(C)

Name

vsh - Invokes the visual shell.

Syntax

vsh *options*

Description

The **vsh** command is a visual command interpreter with an interface much like that of other IBM tools. This shell is recommended for casual users of the system and those with specialized needs that do not include learning the complete operating system. Note that the visual shell can be substituted for a user's normal login shell by altering the **/etc/passwd** file.

See Also

XENIX Visual Shell
sh(C)

WAIT(C)

Name

wait - Awaits completion of background processes.

Syntax

```
wait
```

Description

The **wait** command waits until all background processes started with an ampersand
(&) have finished and reports on abnormal terminations.

See Also

sh(C)

Comment

Not all the processes of a pipeline with three or more stages are children of the shell and thus cannot be waited for.

WALL(C)

Name

wall - Writes to all users.

Syntax

```
/etc/wall
```

Description

The **wall** command reads a message from the standard input until an end-of-file. It then sends this message to all users currently logged in preceded by "Broadcast Message from . . . ". The **wall** command is used to warn all users, as for example, before shutting down the system.

The sender should be super-user to override any protections the users may have invoked.

Files

/dev/tty*

See Also

mesg(C), write(C)

Diagnostics

Cannot send to . . .	The open on a user's **tty** file has failed.

WC(C)

Name

wc - Counts lines, words and characters.

Syntax

```
wc [-lwc] [names]
```

Description

The **wc** command counts lines, words and characters in the named files or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters is to be reported. The default is **-lwc**.

When *names* are specified on the command line, the names are printed along with the counts.

WHAT(C)

Name

what - Identifies files.

Syntax

what *files*

Description

The **what** command searches the given files for all occurrences of the pattern @(#) and prints out what follows until the first tilde (~), greater than sign (>), new-line, backslash (\ ), or null character.

For example, if the shell procedure in file **print** contains:

```
#  @(#)this is the print program
#  @(#)syntax: print [files]
pr $* |lpr
```

then the command:

```
what print
```

displays the name of the file **print** and the identifying strings in that file:

```
print:
      this is the print program
      syntax: print [files]
```

WHO(C)

Name

who - Lists who is on the system.

Syntax

who [*who-file*] **[am I]**

Description

Without an argument, **who** lists the login name, terminal name, and login time for each current XENIX 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 system outages 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 ~ in the place of the device name, and a fossil time indicating 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.

File

/etc/utmp

See Also

utmp(M)

WHODO(C)

Name

whodo - Determines who is doing what.

Syntax

/etc/whodo

Description

The **whodo** command produces merged, reformatted, and dated output from the **who**(C) and **ps**(C) commands.

See Also

ps(C), who(C)

WRITE(C)

Name

write - Writes to another user.

Syntax

write *user* [*tty*]

Description

The **write** command copies lines from your terminal to that of another user. When first called, it sends the message:

```
Message from your-logname your-tty  . . .
```

The recipient of the message should write back at this point. Communication continues until an end-of-file is read from the terminal or an interrupt is sent. At that point, **write** writes:

```
(end of message)
```

on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *tty* argument may be used to indicate the appropriate terminal.

Permission to write may be denied or granted by use of the **mesg**(C) command. At the outset, writing is allowed. Certain commands, in particular **pr**(C), disallow messages to prevent messy output.

If the character ! is found at the beginning of a line, **write** calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using **write**: when you first write to another user, wait for him or her to write back before starting to send. Each party should end each message with a distinctive signal ((**o**) ford"over" is conventional), indicating that the other may reply; (**oo**) for "over and out" is suggested when conversation is to be terminated.

Files

/etc/utmp	To find user
/bin/sh	To execute !

See Also

mail(C), mesg(C), who(C)

XARGS(C)

Name

xargs - Constructs and executes commands.

Syntax

xargs [*flags*] [*command*[*initial-arguments*]]

Description

The **xargs** command combines the fixed *initial-arguments* with arguments read from the standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

The *command* specified, which may be a shell file, is searched for using the shell $PATH variable. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted; characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings, a backslash (\ ) escapes the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (exception: see **-i** flag). Flags **-i**, **-l**, and **-n** determine how arguments are selected for each command invocation. When none of these flags is coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full. Then *command* is executed with the accumulated arguments. This process is repeated until there are no more arguments. When there are flag conflicts (for example, **-l** vs. **-n**), the last flag has precedence.

Values for *flag* are:

-l*number* The specified *command* is executed for each *number* lines of non-empty arguments from the standard input. This is instead of the default single line of input for each *command.* The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first newline unless the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **-x** is forced.

-i*replstr* Insert mode: *command* is executed for each line from the standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr.* A maximum of five arguments in *initial-arguments* may each contain one or more instances of *replstr.* Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **-x** is also forced. **{}** is assumed for *replstr* if not specified.

-n*number* Executes *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments are used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **-x** is also coded, each *number* argument must fit in the *size* limitation or else **xargs** terminates execution.

-t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just before their execution.

-p	Prompt mode: The user is asked whether to execute the specified *command* at each invocation. Trace mode (**-t**) is automatically turned on to print the *command* each time it is expected to execute, followed by a **?** . . . prompt. A reply of **y** (optionally followed by anything) executes the command; anything else, including just a carriage return, skips that particular invocation of *command.*
-x	Causes **xargs** to terminate if any argument list would be greater than *size* characters; **-x** is forced by the options **-i** and **-l**. When neither **-i**, **-l**, or **-n** are coded, the total length of all arguments must be within the *size* limit.
-s*size*	The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **-s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
-e*eofstr*	The *eofstr* is taken as the logical end-of-file string. Underscore (__) is assumed for the logical **EOF** string if -**e** is not coded. The **-e** with no *eofstr* coded turns off the logical **EOF** string capability (underscore is taken literally). The **xargs** command reads the standard input until either end-of-file or the logical **EOF** string is encountered.

The **xargs** command terminates if it either receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see **sh(C)**) with an appropriate value to avoid accidentally returning with **-1.**

Examples

The following moves all files from directory $1 to directory $2, and echoes each **move** command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following combines the output of the parenthesized commands on to one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be printed and prints them one at a time;

```
ls | xargs -p -l lpr
```

or many at a time:

```
ls | xargs -p -l | xargs lpr
```

The following executes **diff**(C) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

YES(C)

Name

yes - Prints string repeatedly.

Syntax

```
yes [string]
```

Description

The **yes** command repeatedly outputs *y* or, if a single string argument is given, *arg* is output repeatedly. The command continues indefinitely unless ended. This is useful in pipes to commands that prompt for input and require a *y* response for a yes. In this case, **yes** terminates when the command it pipes to terminates, so that no infinite loop occurs.

Section 2. Maintenance Commands and Miscellaneous Information

Introduction

This section contains commands and information that can be used to maintain the XENIX system . Included in this section are: commands, descriptions of important files, devices, tables, and programs that are important in maintaining the entire XENIX system.

ALIASES(M)

Name

aliases, aliases.hash, maliases, faliases - Micnet aliasing files.

Description

These files contain the alias definitions for a Micnet network. Aliases are short names or abbreviations that may be used in the **mail**(C) command to refer to specific machines or users in a network. Aliasing allows a complex combination of site, machine, and user names to be represented by a single name.

The **aliases, maliases**, and **faliases** files each define a different type of alias. The **aliases** file defines the standard aliases, which are names for specific systems and users and, in some case, for commands. The **maliases** file defines machine aliases, names, and paths for specific systems. The **faliases** file defines forwarding aliases, which are temporary names for forwarding mail intended for one system or user to another.

The **aliases.hash** file is the hashed version of the aliases file created by the **aliashash**(C) command. The file is used by the **mail**(C) command to resolve all standard aliases and is identical to the aliases file except for a hash table at the beginning of the file. The hash table allows for more efficient access to the entries in the file. The **aliases** file need only be present to generate the **aliases.hash** file. The **aliases** file is not required to run the network.

Each file contains zero or more lines. Each line lists the alias and its meaning. The alias can have up to eight letters and numbers, but must begin with a letter. The meaning can have site, machine, and user login names and other aliases (its exact composition depends on the type of alias). A colon (:) separating the alias and meaning is required.

In the **aliases** file, a line can have the forms:

```
alias:[[site!]machine:] user[,[[site!]machine:]user] . . .
```

```
alias:[[site!]machine:] command-pipeline
```

```
alias:error-message
```

Site and *machine* are the site and machine names of the system to which the user belongs or on which the specified command is to be executed. The site and machine names must end with an exclamation mark (!) or colon (:), respectively, and must be defined in a **systemid** file. A machine alias may be used in place of a site and machine name if it is followed by a colon.

User is a user login name or another alias. User names in a list must be separated by commas. A newline may immediately follow a comma. Spaces and tabs are allowed, but only immediately before or after a comma or newline.

Command-pipeline is any valid command (with necessary arguments) preceded by a pipe symbol (|) and enclosed in double quotation marks. Spaces may separate the command and arguments, but there must be no space between the first double quotation mark and the pipe symbol.

Error-message is any sequence of letters, numbers, and punctuation marks (except a double quotation mark) preceded by a number sign (#) and enclosed in double quotation marks.

In the **faliases** file, each line can have the same form as lines in the **aliases** file except that no more than one user name can be given for any one alias.

In the **maliases** file, a line has the form:

```
alias:[[site!]machine:] . . .
```

Site and *machine* are the site and machine names for a specific network and system. Multiple site and machine names direct messages along the specified path of systems. If no site or machine name is given, the alias is ignored.

Before the **mail** program sends a message, it searches the **aliases.hash**, **faliases**, and **maliases** files to see if any of the names given with the command are aliases. Each file is searched in turn, (**aliases.hash**, **faliases**, then **maliases**) and if a match is found, the alias is replaced with its meaning. If no match is found, the name is assumed to be the valid login name of a user on that machine. The search in the **aliases.hash** file continues until all aliases have been replaced, so it is possible for several replacements to occur for a single name. (If a loop exists, processing continues indefinitely). The **faliases** file is searched once, from beginning to end, even if it contains no aliases. The **maliases** file is searched only if the alias contains a machine alias.

When an alias is a user or a list of users, the **mail** command sends the message to each user in the list. When it is a command-pipeline, the **mail** command starts execution of the command on the specified machine and sends the message as input. When the alias is an error message, the **mail** command ignores the message and instead displays the alias and its meaning on the standard error output.

In all files, any line beginning with a number sign (#) is considered a comment and is ignored.

As a special feature, any alias that contains a site name as the first component of its meaning is automatically prefixed with the machine alias **uucp?**. This alias may be explicitly defined in the **maliases** file to help direct mail between networks to the system performing the uucp link.

Files

/usr/lib/mail/aliases
/usr/lib/mail/aliases.hash
/usr/lib/mail/maliases
/usr/lib/mail/faliases

See Also

aliashash(M), netutil(C), systemid(M), top(M)

ALIASHASH(M)

Name

aliashash - Micnet alias hash table generator

Syntax

/usr/lib/mail/aliashash [**-v**] [**-o** *output-file*] [*input-file*]

Description

The **aliashash** command reads the *input-file* and generates an *output-file* containing a hash table of alias definitions for a Micnet network. The *input-file* must name a file containing alias definitions in the form described for the **aliases** file (see **aliases**(M)). If the **-o** option is not used to specify an *output-file,* the command creates a file with the same name as the *input-file* but with **.hash** appended to it. If no *input-file* is given, the command reads the file named **/usr/lib/mail/aliases** and creates the file named **/usr/lib/mail/aliases.hash.**

If invoked with the **-v** option, the command lists information about the hash table.

The *output-file* contains both the alias definitions given in the *input-file* and the new hash table. The hash table appears at the beginning of the file and is separated from the alias definitions by a blank line. The hash table has three or more lines. The first line is:

```
#<hash>
```

The second line has four entries: the bytes per table entry, the maximum number of items per hash value, the number of entries in the table, and the offset (in bytes) from the beginning of the file to the beginning of the alias definitions.

The next lines (to the end of the hash table) contain the hash table entries. Each line has eight entries (separated by spaces) and each entry has two fields. The first field (one byte) is a checksum (represented as a printable character); the second field is a pointer (in bytes) to the alias definition. The pointer is represented as a hexadecimal number, with leading blanks if necessary, and is always relative to the start of the definitions.

The **aliashash** command is normally invoked by the install option of the **netutil** command. If the alias definitions of a network must be changed, the definitions in the **aliases** file should be changed and a new **aliases.hash** file created using the **aliashash** command. The new **aliases.hash** file must then be copied to all other computers in the network.

Files

/usr/lib/mail/aliashash
/usr/lib/mail/aliases
/usr/lib/mail/aliases.hash

See Also

aliases(M), netutil(C)

Warning: Do not use the **aliashash** file while the network is running. If necessary, create a temporary output file, **aliases.hash-**, using the **-o** option, then type:

```
mv aliases.hash- aliases.hash
```

This will prevent disruption of the network.

ASCII(M)

Name

ascii - Map of the ASCII character set.

Description

The **ascii** map is a map of the ASCII character set. It lists both octal and hexadecimal equivalents of each character. It contains:

OCTAL							
000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

HEXADECIMAL							
00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f -
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

Files

/usr/pub/ascii

BADTRACK(M)

Name

badtrack - Creates bad track table

Description

This facility is used during system installation to build a **badtrack** table on any fixed disk connected to the system. If you add a second fixed disk after installing XENIX, **badtrack** must be run again. Running **badtrack** again will not harm the data on the first disk because only unallocated partitions are scanned. To use the **badtrack** program you must boot the installation diskette.

1. Insert the XENIX installation diskette in drive A and power on the system.

2. At the colon prompt (:), type

```
fd  /etc/badtrack
```

3. Press Enter, and the line

```
Loading
```

displays. When the XENIX installation diskette finishes loading

```
Loaded, press Enter to start
```

is displayed.

4. Press Enter again

The **badtrack** program is loaded from the Installation diskette. The program first displays the number and type of the first fixed disk. The number of each cylinder is displayed as the program proceeds with its tests. After scanning all cylinders on the first disk, **badtrack** displays a list of the bad tracks (listed by cylinder and track). **badtrack** then displays the prompt:

```
     Enter additional bad tracks for drive 0
     Enter cylinder [1-613]
     (press Enter to terminate):
```

If their are no additional bad tracks, press enter. (For the first installation, you do not have to supply additional track information). The program displays a list of the bad tracks, and creates the actual table on the fixed disk. If their are additional bad tracks, at the prompt

```
     Enter cylinder [1-613]
```

Enter the cylinder number followed by Enter. At the prompt

```
     Enter track [0-3]
```

Enter the track number followed by Enter. If you have more bad tracks to report, continue entering cylinder and track data as prompted. After entering the last bad track. Press Enter.

If you have a second fixed disk, **badtrack** repeats the same operation for that disk. After completing all operations, **badtrack** returns control to the XENIX boot program which displays the prompt:

```
XENIX 286 boot

Enter: device program

Press Enter for default: hd  /xenix

:
```

Files

/etc/badtrack

BOOT(M)

Name

boot - XENIX boot program.

Description

The **boot** program is an interactive program used to load and execute standalone XENIX programs. It is used primarily for loading and executing the XENIX kernel, but can be used for any other programs that have been linked for standalone execution. The **boot** program is a required part of the XENIX Operating System and must be present in the root directory of the root file system to ensure successful loading of the XENIX kernel.

The **boot** program is invoked by the system each time the computer is started. A bootstrap loader, loaded by the IBM Personal Computer AT startup ROM, carries out the invocation. A diskette version of **boot** is invoked if the upper diskette drive (drive 0) contains a diskette and block 0 of that diskette contains a valid bootstrap program. A fixed disk version of **boot** is invoked if the diskette drive is empty and block 0 of the active partition of the fixed disk contains a valid bootstrap program. Otherwise, the startup ROM starts ROM BASIC.

When first invoked (or after termination of standalone programs), **boot** prompts for the location of a program to load by displaying the message:

```
XENIX 286 Boot
Enter: device program
Press Enter for default:  hd /xenix
:
```

To specify the location of a program, a device and pathname must be given. The device can be either fd or hd. The pathname must be the full pathname of the file containing the standalone program.

For example, to load /xenix from the diskette, type:

```
fd /xenix
```

Because boot is most often used to load the XENIX kernel from the fixed disk, the default value "hd /xenix" is provided.

If an optional load-address is given, **boot** loads the program at the specified 20-bit physical memory address. This address must be a hexadecimal value in the range 0x800 to 0xF0000. If no address is given, **boot** uses the default load address for the program (given when the program was linked). For example, the default load address for the XENIX kernel is 0. Addresses in the ranges 0 to 0x800 and 0xF0000 to 0xFFFFF are reserved for the system's interrupt tables and the **boot** program, respectively. Interrupt table addresses can be used if the program operates in protected mode. The **boot** addresses can be used if the program does not return control to **boot** after terminating.

You can display a list of the current allowable device names by typing the question mark (?).

Diagnostics

On an error, **boot** displays an error message, then returns to its prompt. The following is a list of the most common messages:

bad magic number
The given file is not an executable program.

bad runtime environment
The given program is not a standalone program. The program must be a small or middle model program, and the absolute flag of the program header must be set.

pathname not found
The supplied pathname does not correspond to an existing file.

Stage 1 boot failure

The bootstrap loader cannot not find or read **boot**. You must restart the computer and supply a file system disk with **boot** in the root directory.

Unknown device: device

Only fd and hd can be used for device names.

Files

/boot

See Also

hd(M)

Comments

The **boot** program cannot be used to load programs that have not been linked for standalone execution. To create standalone programs, the **-A** option of the XENIX linker must be used.

Although standalone programs can operate in real or protected mode, they must not be large or huge model programs. Programs in real mode can use the input/output routines of the IBM Personal Computer AT startup ROM. Programs that wish to return control to **boot** must be linked with the library **/usr/sys/boot/lib_stand.**

CLOCK(M)

Name

clock - The system real-time clock.

Description

The **clock** file provides access to the battery-powered, real-time clock of the IBM Personal Computer AT. Reading this file returns the current time; writing to the file sets the current time. The time, 10 bytes long, has the following form:

```
MMddhhmmyy
```

where *MM* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *yy* is the last two digits of the year. For example, the time:

```
0122140884
```

is 14:08 on January 22, 1984.

File

/dev/clock

See Also

setclock(M)

CMOS(M)

Name

cmos - Displays and sets the configuration data base.

Syntax

cmos [*address* [*value*]]

Description

The **cmos** command displays and/or sets the values in the CMOS configuration data base. This battery-powered data base stores configuration information about the computer that is used at power up to define the system hardware configuration and to direct boot procedures. The data base is 64 bytes long and is reserved for system operation (see the *IBM Personal Computer AT Technical Reference Manual* for details).

The **cmos** command is typically used to alter the current hardware configuration when new devices are added to the system. When only *address* is given, the command displays the value at that address. If both and *address* and a *value* are given, the command assigns the value to that address. If no arguments are given, the command displays the entire contents of the data base.

The CMOS configuration data base may also be examined and modified by reading from and writing to the **/dev/cmos** file. Because successful system operation depends on correct configuration information, the data base should be modified by experienced system administrators only.

Files

/etc/cmos
/dev/cmos

CONSOLE(M)

Name

console, color, monochrome - The standard system terminal.

Description

The **console** file provides access to the keyboard and the current display device of the IBM Personal Computer AT and is the default device for all system error messages and for interaction during the system boot sequence. The file has the same input and output characteristics as the terminal devices described in **tty(M)**.

The **console** file is actually two files in one: a readable file to the keyboard and a writable file to one of two possible display devices, color and monochrome. The color file provides access to a color/graphics adapter, the monochrome file to the monochrome adapter. Only one file is used as the current display device. It is defined by the hardware configuration in the CMOS configuration data base (see **cmos**(M)) and assigned when the system is started. The other file remains available to be written to; neither file can be read. It is an error to attempt to access the color or monochrome file when no corresponding adapter exists or when the file is the current display device.

The following is a list of the control sequences supported by the IBM Personal Computer AT display devices. Note that "ESC" is the ASCII escape character, 027, and "Pn" is a numeric parameter to be supplied. Although spaces are shown between characters in a sequence, they are not allowed when actually forming the sequence.

ANSI Mnemonic	Sequence	Action
CPL	ESC [Pn F	Move cursor to previous line
CNL	ESC [Pn E	Move cursor to next line
CUB	ESC [Pn D	Move cursor backward
CUF	ESC [Pn C	Move cursor forward
CUU	ESC [Pn A	Move cursor up
CUD	ESC [Pn B	Move cursor down
CUP	ESC [Pm ; Pn H	Move cursor to row Pm, column Pn
DCH	ESC [Pn P	Delete character
DL	ESC [Pn M	Delete line
ECH	ESC [Pn X	Erase character
ED	ESC [Pn J	Erase display *
EL	ESC [Pn K	Erase line *
ICH	ESC [Pn @	Insert char
IL	ESC [Pn L	Insert line
SGR	ESC [Pn m	Select graphic rendition **

* For the ED and EL sequences, if Pn is:

0 Erase from start of screen or line to cursor

1 Erase from cursor to end of the screen or line

2 Erase entire screen or line

** For the SGR sequence, if Pn is:

0 Enter normal video mode

7 Enter inverse video mode

For all other sequences, Pn defines the number of times to repeat the given action. The default count is 1.

The special character sequence:

```
ESC Q Pd Pc Ps Pc
```

sets the output of the function key given by Pd to the string Ps. The key number Pd must be one less than the actual function key number. The string Ps must be enclosed by the quoting character Pc, and must not contain any occurrence of Pc. For example, the sequence:

```
ESC Q 0 "date"
```

sets the output of function key 1 to "date". The escape sequence:

```
^ Pn
```

may be used in the string Ps to represent an ASCII control character (0x0 to 0x20). Pn must be a character in the ASCII range 0x20 to 0x40. For example, the escape sequence "^*" represents the newline character Ctrl-J. A caret "^" in Ps must be given as "^^".

Files

/dev/console
/dev/color
/dev/monochrome

See Also

tty(M)

DAEMON.MN(M)

Name

daemon.mn - Micnet mailer daemon

Syntax

/usr/lib/mail/daemon.mn [**-ex**]

Description

The mailer **daemon** performs the "backend" networking functions of the **mail** , **rcp** , and **remote** commands by establishing and servicing the serial communications link between computers in a Micnet network.

When invoked, the **daemon** creates multiple copies of itself, one copy for each serial line in the network. Each copy opens the serial line, creates a startup message for the LOG file corresponding to that line, and waits for a response from the daemon at the other end. The startup message lists the names of the machines to be connected, the serial line to be used, and the current date and time. If the **daemon** receives a correct response, it establishes the serial link and adds the message "first handshake complete" to the LOG file. If there is no response, the **daemon** waits indefinitely.

If invoked with the **-x** option, the **daemon** records each transmission in the LOG file. A transmission entry shows the direction of the transmission (tx for transmit, rx for receive), the number of bytes transmitted, the elapsed time for the transmission (in minutes and seconds), and the time of day of the transmission (in hours, minutes, and seconds). Each entry has the form:

```
direction byte__count elapsed__time @ time__of__day
```

The **daemon** also records the date and time every hour. The date and time have the same format as described for the **date** (C) conunand.

If invoked with the **-e** option, the **daemon** records all transmission errors in the LOG file. An error entry shows the cause of the error preceded by the name of the **daemon** subroutine that detected the error.

The mailer **daemon** is normally invoked by the start option of the **netutil**(C) command and is stopped by the stop option.

During the normal course of execution, the mailer **daemon** uses several files in the **/user/spool/micnet/remote** directory. These files provide storage for LOG entries, commands issued by the **remote**(C) command, and a list of processes under **daemon**

Files

/usr/lib/mail/daemon.mn
/usr/spool/micnet/remote/*/LOG
/usr/spool/micnet/remote/*/mn*
/usr/spool/micnet/remote/local/mn*
/usr/spool/micnet/remote/lock
/usr/spool/micnet/remote/pids

See Also

netutil(C)

DEFAULT(M)

Name

default - Default program information directory.

Description

The files in the directory **/etc/default** contain the default information used by system commands such as **backup**(C) and **remote**(C). Default information is any information required by the command that is not explicitly given when the command is invoked.

The directory may contain zero or more files. Each file corresponds to one or more commands. A command searches a file whenever it has been invoked without sufficient information. Each file contains zero or more entries that define the default information. Each entry has the form:

keyword

or

keyword=value

where *keyword* identifies the type of information available and *value* defines its value. Both *keyword* and *value* must consist of letters, digits, and punctuation. The exact spelling of a *keyword* and the appropriate *values* depend on the command and are described with the individual commands.

Any line in a file beginning with a number sign (#) is considered a comment and is ignored.

Files

/etc/default/backup
/etc/default/cron
/etc/default/dos
/etc/default/dumpdir
/etc/default/lpd
/etc/default/micnet
/etc/default/mkuser
/etc/default/msdos
/etc/default/passwd
/etc/default/restor
/etc/default/su

See Also

cron(C), dos(C), backup(C), dumpdir(C), lpr(C), mkuser(C), pwadmin(C), remote(C), restor(C), su(C)

ENVIRON(M)

Name

environ - The user environment.

Description

The user environment is a collection of information about a user, such as his login directory, mailbox, and terminal type. The environment is stored in special "environment variables," which can be assigned character values, such as names of files, directories, and terminals. These variables are automatically made available to programs and commands invoked by the user. The commands can then use the values to access the user's files and terminal.

The following is a partial list of environment variables:

PATH	Defines the search path for the directories containing commands. The system searches these directories whenever a user types a command without giving a full pathname. The search path is one or more directory names separated by colons (:). Initially, PATH is set to :/bin:/usr/bin.
HOME	Names the user's login directory. Initially, HOME is set to the login directory given in the user's **passwd** file entry.
TERM	Defines the type of terminal being used. This information is used by commands such as **more**(C), that rely on information about the capabilities of the user's terminal. The variable may be set to any valid terminal name (see **terminals**(M)) directly or by using the **tset**(C) command.

TZ Defines time zone information. This information is used by **date**(C) to display the appropriate time. The variable may have any value of the form *xxxnzzz* where *xxx* is standard local time zone abbreviation, *n* is the difference in hours from GMT, and *zzz* is the daylight-saving local time zone abbreviation (if any). For example: EST5EDT. The difference for a location east of England can be given as a negative number.

The environment can be changed by assigning a new value to a variable. An assignment has the form:

name=value

For example, the assignment:

```
TERM=h29
```

sets the **TERM** variable to the value .h29. The new value can be exported to each subsequent invocation of a shell by exporting the variable with the **export** command (see **sh**(C)) or by using the **env**(C) command.

A user may also add variables to the environment, but must be sure that the new names do not conflict with exported shell variables such as MAIL, PS1, PS2, and IFS. Placing assignments in the **.profile** file is a useful way to change the environment automatically before a session begins.

Note that the environment is made available to all programs as a string of arrays. Each string has the form:

name=value

where the *name* is the name of an exported variable and the *value* is the variable's current value.

See Also

env(C), login(M), sh(C), profile(M)

FD(M)

Name

fd∅, fd1 - Diskette drive devices.

Description

The *fd∅* and *fd1* files provide access to the standard diskette drives. Each file corresponds to one diskette drive and may only be accessed if the corresponding drive is present. The file *fd∅* provides access to the boot diskette drive (or drive ∅), and *fd1* provides access to the optional secondary drive (drive 1). The files are typically used to mount file systems based on diskette (see **mount**(C)) and to create backup copies of files stored on fixed disk (see **tar**(C) and **backup**(C)).

To implement the standard diskette drive files, the *fd∅* and *fd1* files are actually linked to two of 10 underlying device files. These underlying files correspond to the 10 possible combinations of diskette drives and disk formats. Each file defines a specific drive, density, sector count, side count, and capacity as shown by the following table.

Name	Drive	Density (TPI)	Sectors	Sides	Capacity (KBytes)
fd∅48ss8	0	48	8	1	160
fd∅48ds8	0	48	8	2	320
fd∅48ss9	0	48	9	1	180
fd∅48	0	48	9	2	360
fd∅48ds9	0	48	9	2	360
fd148ss8	1	48	8	1	160
fd148ds8	1	48	8	2	320
fd148ss9	1	48	9	1	180
fd148	1	48	9	2	360
fd148ds9	1	48	9	2	360
fd∅96ds15	0	96	15	2	1200
fd196ds15	1	96	15	2	1200

The *fd∅* and *fd1* files are linked to the device files that define the format of the resident diskette fd∅96ds15 and fd196ds15. The drives can be accessed through *fd∅* and *fd1* or directly through the corresponding format-specific files.

The *rfd∅* and *rfd1* files are the raw (or character I/O) files associated with the standard diskette drives. There are similar character I/O files for the underlying format-specific files. These files are used by the **format(M)** command, and can also be used by special applications. When accessing the character I/O files, input and output must be aligned on 512 byte boundaries. One method to ensure this is to create a buffer that contains 512 bytes more than actually needed, then round the start address up to the next multiple of 512.

Attempting to access a drive of one format through a device file of a different format may cause unpredictable results or errors. Furthermore, attempting to access a disk of one format in a drive of another may have unpredictable results or errors. Attempting to read a nine sector disk as an eight sector disk causes unpredictable results.

The **format(M)** Command may be used to format diskettes.

It should be noted that the *DOS* file utilities additionally support alias device names according to the following table:

Alias	Names
A	fd048ds9, fd048ds8, fd048ss9, fd048ss8
B	fd148ds9, fd148ds8, fd148ss9, fd148ss8
X	fd096ds15
Y	fd196ds15

Note

These alias device names are only supported for *DOS* file utilities.

Files

Block I/O files

```
/dev/fdØ              /dev/fd1
/dev/fdØ48ss8         /dev/fd148ss8
/dev/fdØ48ds8         /dev/fd148ds8
/dev/fdØ48ss9         /dev/fd148ss9
/dev/fdØ48ds9         /dev/fd148ds9
/dev/fdØ96ds15        /dev/fd196ds15
/dev/fdØ48            /dev/fd148
```

Character I/O files

```
/dev/rfdØ             /dev/rfd1
/dev/rfdØ48ss8        /dev/rfd148ss8
/dev/rfdØ48ds8        /dev/rfd148ds8
/dev/rfdØ48ss9        /dev/rfd148ss9
/dev/rfdØ48ds9        /dev/rfd148ds9
/dev/rfdØ96ds15       /dev/rfd196ds15
```

See Also

format(M)

FDISK(M)

Name

fdisk - Creates disk partitions.

Syntax

```
fdisk [option]specialfile
```

Description

The **fdisk** program interactively creates a disk partition table for the fixed disk drive given by *specialfile*. The partition table defines the location and size of up to four disk partitions on the given fixed disk. One of the four partitions is reserved for the **Badtrack Table(BTT)**. The **fdisk** program prompts for information about the disk partitions, constructs a partition table, then copies the table to a reserved location on the first block of the fixed disk. Each partition provides distinct storage space on the fixed disk that is not accessible through the other partitions. With the following option it is possible to use the **fdisk** program in other than an interactive fashion:

- p [*file*] Provides for the creation of a protofile. This file contains the necessary information for the **fdisk** command to function noninteractively. For each partition to be created a single line is added to the protofile. This line contains three fields, separated by spaces. The first field is the length of the partition in blocks, the second contains the character 'a' if the partition is to be active and some other character if it is not, the third field contains a number indicating the type of the partition.

The **fdisk** program must be invoked with the name of the raw device special file corresponding to an entire fixed disk (see **hd**(M)). The program displays a message and a prompt (*), and waits for commands. The program has the commands: **a** (activate), **c** (create), **d** (delete), **?** (help), **p** (print), **q** (quit), and **w** (write).

The **a** command activates a partition, naming it as the partition from which to load the operating system. The command prompts for the partition number.

The **c** command creates a disk partition by prompting for the partition's number (1 through 4), starting block, size (in blocks), and type (XENIX or DOS). No more than four partitions can be created. The partitions must not overlap and their size must not be greater than the available space on the fixed disk. The type must define the type of file system or files to be stored in the partition. A partition that already exists must be deleted before it can be recreated.

The **d** command deletes a disk partition by prompting you for the number of the partition (1 through 4). The command prompts for a partition number.

The **?** help command displays a list of the **fdisk** commands.

The **p** command prints information about the partition table currently being created. The information includes the partition number, its activation state, type, starting and ending cylinders numbers, and size in sectors. Only partitions that have been created using **c** are shown. If a partition is active, the letter A is given in the Active column; otherwise N is given.

The **q** command stops the **fdisk** program and returns control to the shell.

The **w** command copies the partition table to the fixed disk, destroying any current table on that disk.

Files

/etc/fdisk

See Also

hd(M)

Comment

If an attempt is made to write a partition table to a disk before activating a partition, **fdisk** will display a warning before proceeding with the operation. The **fdisk** program will not allow updating of the partition table on an active disk. To modify the partition table you must load the /xenix.fd diskette and type

```
fd /xenix.fd
```

at the boot program prompt **(:)** **fdisk** can be run. Remember to run **haltsys** before removing the installation diskette.

FORMAT(M)

Name

format - Formats diskettes.

Syntax

format *specialfile*

Description

The **format** command formats the diskette in the diskette drive given by *specialfile.* The *specialfile* must be the name of the character I/O device special file that corresponds to the diskette drive (see **fd**(M)). The diskette is given a format compatible with the given drive. For example, the command:

```
format /dev/rfdØ
```

formats the diskette in the drive 0. Formatting the diskette destroys any data previously stored on it.

> **Warning:** When you are working on a terminal connected to a IBM Personal Computer AT, the **format** command formats the diskette in the system unit. Make certain the diskette you intend to format is in the drive on the system unit.

The diskette must be in the drive before invoking the command.

Files

/bin/format

See Also

fd(M)

GETTY(M)

Name

getty - Sets terminal mode.

Syntax

```
/etc/getty [char]
```

Description

The **getty** program is invoked by **init(M)** immediately after a terminal is opened for user logins. The **getty** program displays a "login:" message, then waits for the user to type a login name. While reading the name, **getty** attempts to adapt the system to the speed and type of terminal being used. Once the name is read, **getty** calls **login(M)** with the login name as the argument.

The **init** program calls **getty** with a single character argument taken from the **ttys(M)** file entry for the terminal line. This argument determines the line speed for the terminal and also the "login:" greeting message, which can contain control characters to initialize the terminal for proper communication.

If the user types a name and terminates it with a newline (ASCII LF) or carriage return (ASCII CR), **getty** scans the name for uppercase alphabetic characters. If only uppercase characters are found, **getty** adapts the system to map all subsequent lowercase characters into the corresponding uppercase characters. Furthermore, if the name terminates with a carriage return character, **getty** sets the terminal's serial line mode to CRMOD.

If, on the other hand, the user presses the Interrupt (Del) key, **getty** writes the login message again. It also changes the serial line speed if *char* is 1 or 3 as described below. This allows the system to adapt to terminals whose line speeds vary.

After a name has been typed and scanned, **getty** passes it to **login**(M), which asks for the user's password and completes the login process.

The following arguments from the **ttys** file are understood:

-	110 baud. Intended for an ASR-33 console, for example, an operator's console.
0	150 baud for an ASR-37 console.
1	Cycles through 300-150-110-1200 baud. Useful for dial-up lines.
2	300-baud console decwriter.
3	Cycles through 1200-300-150-110 baud. Recommended for dialup lines.
4	2400 baud.
5	4800 baud.
6	9600 baud.
7	9600 baud for IBM 3101 terminal.

See Also

login(M), ttys(M), init(M)

GROUP(M)

Name

group - Format of the group file.

Description

The **group** file contains for each group the following information:

- Group name
- Encrypted password (optional)
- Numerical group ID
- Comma-separated list of all users allowed in the group.

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a newline. If the password field is null, no password is demanded.

This file resides in directory **/etc**. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group IDs to names.

Files

/etc/group

See Also

newgrp(C), passwd(C), passwd(**M**)

HD(M)

Name

hdØØ, hd1Ø - Fixed disk devices.

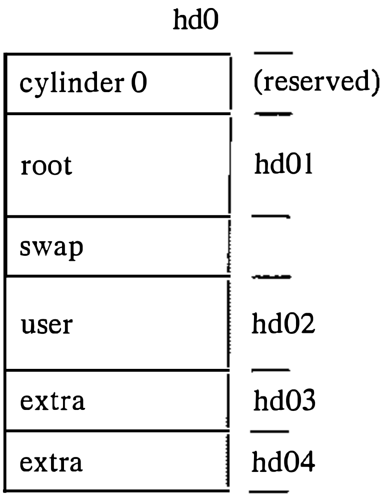
Description

The *hdØØ* and *hd1Ø* files provide access to the fixed disk drives of the IBM Personal Computer AT computer. Each file corresponds to one fixed disk drive and may be accessed if the corresponding fixed disk drive is present. The *hdØØ* file corresponds to the standard fixed disk drive, and *hd1Ø* corresponds to an optional or second fixed disk. Each file permits access to the entire contents of the given fixed disk, regardless of partition boundaries.

The *hdØ1*, *hdØ2*, *hdØ3*, and *hdØ4* files provide access to each of the four possible partitions on the standard fixed disk. Similarly, the *hd11*, *hd12*, *hd13*, and *hd14* provide access to the partitions of an optional or second disk. Each file corresponds to a region on the given fixed disk that has been specifically defined as a partition using the **fdisk**(M) program. A partition file may only be accessed if the corresponding partition has been defined. Each file permits access to the contents of the given partition only.

The *hdØa* and *hd1a* files provide access to the active partitions of the respective fixed disk drives. Only one of four partitions on the fixed disk can be the active partition. This partition usually contains the root file system for the system, including the boot program used to load XENIX. The active partition can be set using the **fdisk**(M) program.

The following illustrates a typical layout:



The *rhd∅∅* and *rhd1∅* files are the raw (or character I/O) device files associated with the standard and optional fixed disk drives. The raw files are used by the **fdisk**(M) program.

Because system operation depends on the content of these blocks, they should be modified by experienced users only. Each block contains 1024 bytes.

Files

Block I/O files

/dev/hd00/dev/hd01
/dev/hd02
/dev/hd03
/dev/hd04
/dev/hd10/dev/hd11
/dev/hd12
/dev/hd13
/dev/hd14

Character I/O files

/dev/rhd00/dev/rhd01
/dev/rhd02
/dev/rhd03
/dev/rhd04
/dev/rhd10/dev/rhd11
/dev/rhd12
/dev/rhd13
/dev/rhd14

See Also

fdisk(M)

INIT(M)

Name

init - Process control initialization.

Syntax

```
/etc/init
```

Description

The **init** program is invoked as the last step of the boot procedure and as the first step in enabling terminals for user logins. This is one of three programs (**init**, **getty**(M), and **login**(M)) used to initialize a system for execution.

The **init** program creates a process for each terminal on which a user may log in. It begins by opening the console device, **/dev/console**, for reading and writing. It then invokes a shell, which asks for a password to start the system in maintenance mode. You can type the password or terminate the shell by typing ASCII end-of-file (Ctrl-D) at the console. If the shell terminates, **init** performs several steps to begin normal operation. It invokes a shell and reads the commands in the **/etc/rc** file. This command file performs housekeeping tasks such as removing temporary files, mounting file systems, and starting daemons. Then **init** reads the file **/etc/ttys** and forks several times to create a process for each terminal device in the file. Each line in the **/etc/ttys** lists the state of the line (0 for closed, 1 for open), the line mode, and the serial line (see **ttys**(M)). Each process opens the appropriate serial line for reading and writing, assigning the file descriptors 0, 1, and 2 (standard input, output, and error file) to the line. If the serial line is connected to a modem, the process delays opening the line until someone has dialed up and a carrier has been established on the line.

Once **init** has opened a line, it executes the **getty** program, passing the line mode as an argument. The **getty** program reads the user's name and invokes **login**(M) to complete the login process (see **getty**(M) for details). The **init** program waits until the user logs out by typing ASCII end-of-file (Ctrl-D) or by hanging up. It responds by waking up and removing the former user's login entry from the file **/etc/utmp**, which records current users, and makes a new entry in the file **/usr/adm/wtmp**, which is a history of logins and logouts. Then the corresponding line is reopened, and **getty** is reinvoked.

The **init** program has special responses to the hangup, interrupt, and quit signals. The hangup signal SIGHUP causes **init** to change the system from normal operation to maintenance mode. The interrupt signal SIGINT causes **init** to read the **ttys** file again to open any new lines and close lines that have been removed. The quit signal SIGQUIT causes **init** to disallow any further logins. In general, these signals have a significant effect on the system and should not be used by an inexperienced user. Instead, similar functions can be safely performed with the **enable**(C), **disable**(C), and **shutdown**(C) commands.

Files

/dev/tty
/etc/utmp
/usr/adm/wtmp
/etc/ttys
/etc/rc

See Also

disable(C), enable(C), login(M), kill(C), sh(C), shutdown(C), ttys(M), getty(M)

LD(M)

Name

ld - Invokes the link editor.

Syntax

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

Description

The **ld** command combines several object programs into one, resolves external references, and searches libraries. It is strongly suggested that **ld** be invoked only through **cc**; **cc** invokes **ld** with all the necessary C-language support routines. If you choose to use **ld** directly and fail to get all of the components in the right places, your resultant file will fail to execute. Furthermore, many of the options available on "traditional" XENIX booters are not supported in the same form with this version of **ld**.

In the simplest case, several object *files* are given, and **ld** combines them, producing an executable object module. The output of **ld** is left in the file **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 an argument is a library, it must be preceded by the **-l** switch, and it must have been processed by **ranlib**. The first member of a library is named '_____.SYMDEF', which is understood to be a dictionary for the library. The dictionary is searched iteratively to satisfy as many references as possible. A library is processed at the point it is encountered in the argument list. Only routines defining an unresolved external reference are loaded.

The **ld** command understands several options, which should normally appear before the names of the object files. Exceptions are libraries, which should be specified with the **-l** *after* all object files have been given. Options are described below:

-o	The *name* argument after **-o** is used as the name of the **ld** output file, instead of **a.out.**
-C	Case is significant in symbol names. This option should always be used for C programs because case is significant in C.
-S	No symbol table **.sym** file is created. If this flag is not specified, then a **.sym** file is created containing global symbols only.
-l *libname*	Search *libname* for unresolved external references. The **-l** switch may be used multiple times on a command line, but only one library name may follow each switch. A library is searched when its name is encountered, so the placement of a **-l** is significant. Note that *libname* is a full pathname and not a name to be searched for in a default directory.
-i	Specify separate instruction and data spaces: when the output file is executed, the program text and data areas live in separate address spaces. The text portion is read-only and is shared by all users executing the file.
-m	The following argument, *mapfile*, is taken to be the name of the file in which a map of external and local variables is placed, along with memory allocation information.

-M *string*	Sets the program configuration. This configuration defines the programs memory model, word order, data threshold, and enables C language enhancements such as advanced instruction set and keywords. The *string* may be any combination of the following (the **s**, **m**, and **l** are mutually exclusive):

s Creates a small model program (default).

If the **-M** switch is not given. Small model is used and variables are stored as segment offsets. In the default case, the following object modules and libraries are used:

Object modules:	/lib/ScrtØ.o or /lib/SmcrtØ.o
Libraries:	/lib/Slibcfp.a and /lib/Slibc.a

m Creates a middle model program.

If this switch is given, middle model is used, and text addresses contain both a segment and an offset. Data addresses contain only an offset into the data segment. This means that program text may be larger then 64K-bytes, but program data size is limited to this figure. The **-Mm** option implies **-i**. To support middle model, the following object modules and libraries are used:

Object modules:	/lib/McrtØ.o or /lib/McrtØ.o
Libraries:	/lib/Mlibcfp.a and /lib/Mlibc.a

l Creates a large model program.

If this switch is given, large model is used, and text addresses contain both a segment and an offset. This means that program text and data size may be larger than 64K-bytes. The **-Ml** option implies **-i**. To support large model, the following object modules and libraries are used:

Object modules:	/lib/LcrtØ.o or /lib/LmcrtØ.o
Libraries:	/lib/LlibcFp.a and /lib/Llib:bc.a.br

The **cc** command normally calls **ld** with the following arguments, where *X* below is either S, M, or L depending on whether the **-M** flag was specified and *files* are the user's object files and libraries:

```
/lib/txtdgrp.o  [if not -i,-Mm, or -Ml]
/lib/seg.o
/lib/XcrtØ.o  [ /lib/XmcrtØ.o if -p ]
-C
-S  [ if not -s ]
-M  [ if -M]
-L  [ if -Ml]
-o outfile
-m map  [ if -m ]
```

files

/lib/Xlibcfp.a
/lib/Xlibc.a

Files

/lib/lib*.a	Libraries
/usr/lib/lib*.a	More libraries
/usr/?lib/lib*.a	More libraries
/lib/txtdgrp.o	Needed if no separate I & D
/lib/seg.o	Always needed for C programs
/lib/McrtØ.o	Middle model runtime initialization
/lib/Mlibcfp.a	Middle model
/lib/Mlibc.a	Middle model
/lib/MmcrtØ.o	Middle model runtime initialization with profiling
/lib/LcrtØ.o	Large model runtime initialization
/lib/Llibcfp.a	Large model
/lib/Llibc.a	Large model
/lib/LmcrtØ.o	Large model runtime initialization with profiling
/lib/ScrtØ.o	Small model runtime initialization
/lib/SmcrtØ.o	Small model runtime initialization with profiling
/lib/Slibcfp.a	Small model
/lib/Slibc.a	Small model
a.out	Output file

LOGIN(M)

Name

login - Gives access to the system.

Description

The **login** program is used at the beginning of each terminal session to identify the user attempting to log in to the system. The **login** program is invoked by **getty**(M), after a login name has been typed in response to the "login:" message. The **login** message prompts for the user's password, then turns off echoing (where possible) to prevent the password from appearing on the terminal screen when typed.

Once a password has been typed, **login** encrypts the password and compares it with the encrypted password in the user's password entry (see **passwd**(M)). If there is a match, the login is successful.

If password aging is in effect and the current password is out of date, the **passwd**(C) command is automatically invoked. In this case, the user must change the password, then attempt to log in again.

If the login is not completed within a certain period of time (for example, one minute), **login** either returns control to **getty** or disconnects the dial-up line without outputting any message.

After a successful login, **login** updates accounting files, displays a message about the existence of any mail, then executes the start-up profile file, **/etc/profile**, and the user's **.profile** file in his home directory. (see **profile**(M)). The **login** program then initializes the user and group IDs and the working directory. Finally, it executes a command interpreter (usually **sh**(C)) according to specifications found in the **/etc/passwd** file.

At some installations, an option may be invoked that requires a second external password to be entered. This occurs only for dial-up connections and is prompted by the message "External security:". Both passwords are required for a successful login.

When the user's shell is finally invoked, argument 0 of the command interpreter is a dash (-) followed by the last component of the interpreter's pathname. The *environment* (see **environ**(M)) is initialized to:

HOME= your-login-directory

PATH=:/bin:/usr/bin

The user's file creation mask is set to octal 022. See **umask**(C) for more information.

Files

/etc/utmp	Accounting
/usr/adm/wtmp	Accounting
/usr/spool/mail/*your-name*	Your mailbox
/etc/motd	Message of the day
/etc/passwd	Password file
/etc/profile	System profile
$HOME/.profile	Personal profile

See Also

mail(C), newgrp(C), sh(C), passwd(C), su(C), umask(C), passwd(M), profile(M), environ(M), getty(M)

Diagnostics

Login incorrect
The user name or the password is incorrect.

No shell, cannot open password file, no directory
Your account has not been properly set up.

Your password has expired. Choose a new one.
Password aging is implemented and yours has expired.

LP(M)

Name

lp, lp0, lp1, lp2 - Line printer device interfaces.

Description

The **lp**, **lp0**, **lp1**, **lp1**, and **lp2** files provide access to the optional parallel ports of the IBM Personal Computer AT. The **lp0** and **lp1** files provide access to parallel ports 1 and 2, respectively. The **lp2** file provides access to the parallel port of a monochrome and printer adapter. The **lp** file is actually a link to either **lp0**, **lp1**, or **lp2**, and is the output file for all **lpr**(C) commands. Because the files are intended to give access to a standard dot matrix printer, all bytes written to a file are passed directly to the given port without translation.

Files

/dev/lp
/dev/lp0
/dev/lp1
/dev/lp2

See Also

lpr(C)

MEM(M)

Name

mem, kmem - Memory image file.

Description

The **mem** file provides access to the computer's physical memory. All byte addresses in the file are interpreted as memory addresses. Thus, memory locations can be examined in the same way as individual bytes in a file. Note that accessing a nonexistent location causes an error.

The **kmem** file is the same as **mem** except that it corresponds to kernel virtual memory rather than physical memory.

In rare cases, the **mem** and **kmem** files may be used to write to memory and memory-mapped devices. Such patching is not intended for the inexperienced user and may lead to a system failure if not conducted properly. Patching device registers is likely to lead to unexpected results if the device has read-only or write-only bits.

Files

/dev/mem
/dev/kmem

MESSAGES(M)

Name

messages - Description of system console messages.

Description

This section describes the various system messages that may appear on the system console. The messages are categorized as follows:

Catastrophic	Recovery is impossible.
System inconsistency	A contradiction exists in the kernel.
Abnormal	A probably legitimate but extreme situation exists.
Hardware	Indicates a hardware problem.

Serious system messages begin with "panic" and indicate hardware problems or kernel inconsistencies that are too big for continued operation. After displaying the message, the system stops. Reloading is required.

System inconsistency messages indicate problems usually traceable to hardware malfunction, such as memory failure. These messages rarely occur since associated hardware problems are generally detected before such an inconsistency occurs.

Abnormal messages represent kernel operation problems, such as the overflow of critical tables. It takes extreme situation to bring these problems about, so they should never occur in normal system use.

Hardware messages normally specify the device, *dev*, that caused the error. Each message gives a device specification of the form *nn/mm* where *nn* is the major number of the device, and *mm* is its minor number. The command pipeline:

```
ls -l /dev | grep
```

nn | grep *mm*

may be used to list the name of the device associated with the given major and minor numbers.

System Messages

**** ABNORMAL System Shutdown ****

This message appears when errors occur during normal system shutdown. It is usually accompanied by other system messages. *System inconsistency, unrecoverable.*

bad block on dev *nn/mm dev*

A nonexistent disk block was found on, or is being inserted in, the structure's free list. *System inconsistency.*

bad count on dev *nn/mm*

A structural inconsistency in the superblock of a file system. The system attempts a repair, but this message probably is followed by more error messages about this file system. *System inconsistency.*

bad free count on dev *nn/mm*

A structural inconsistency in the superblock of a file system. The system attempts a repair, but this message will probably be followed by more error messages about this file system. *System inconsistency.*

bad signature (*xx*) on drive *num*

The fixed disk drive specified by *num* has not been properly initialized. The drive should not be used until **fdisk**(M) has been used to initialize the drive. *Hardware.*

can't read bad block list on drive *num*

An attempt to read bad block information from the specified fixed disk has failed. This may indicate a hardware malfunction. *Hardware.*

err on dev *name* (*nn/mm*)

This is the way that most device driver diagnostic messages start. The message indicates the specific driver and error message. The *name* is a word identifying the device. *Hardware.*

iaddress > 2^24

This indicates an attempted reference to an illegal block number, one so large that it could only occur on a file system larger than eight billion bytes. *Abnormal.*

inode table overflow

Each open file requires an inode entry to be kept in memory. When this table overflows, the specific request is refused. Although not catastrophic to the system, this event may damage the operation of various spoolers, daemons, the mailer, and other important utilities. Anomalous results and missing data files are a common result. *Abnormal.*

interrupt from unknown device, vec=*xxxx*

The CPU received an interrupt via a supposedly unused vector. This message is followed by "panic: unknown interrupt." Typically this event comes about when a hardware failure miscomputes the vector of a valid interrupt. *Hardware.*

no file

There are too many open files; the system has run out of entries in its "open file" table. The warnings given for the message "inode table overflow" apply here. *Abnormal.*

no space on dev *mm/mm dev*

This message means that the specified file system has run out of free blocks. Although not normally as serious, the warnings discussed for "inode table overflow" apply: often programs are written casually and ignore the error code returned when they tried to write to the disk; this results in missing data and "holes" in data files. The system administrator should keep close watch on the amount of free disk space and take steps to avoid this situation. *Abnormal.*

**** Normal System Shutdown ****

This message appears when the system has been shutdown properly. It indicates that the machine may now be reloaded or powered down.

out of inodes on dev *nn/mm*

The indicated file system has run out of free inodes. The number of inodes available on a file system is determined when **mkfs**(C) is run. The default number is quite generous; this message should be very rare. The only recourse is to remove some unused files from that file system, or backup the entire system to a backup device, rerun **mkfs**(C) with more inodes specified, and restore the files from backup. *Abnormal.*

out of text

When programs linked with the **ld -i** or **-n** switch are run, a table entry is made so that only one copy of the pure text will be in memory even if there are multiple copies of the program running. This message appears when this table is full. The system refuses to run the program that caused the overflow. Note that there is only one entry in this table for each different pure text program. Multiple copies of one program will not require multiple table entries. Each "sticky" program (see **chmod**(C)) requires a permanent entry in this table; nonsticky pure text programs require an entry only when there is at least one copy being executed. *Abnormal.*

panic: bad 287 int

Attempted execution of real mode 287 instruction. *System inconsistency, irrecoverable.*

panic: blkdev

An internal disk I/O request, already verified as valid, is discovered to be referring to a nonexistent disk. *System inconsistency, irrecoverable.*

panic: devtab

An internal disk I/O request, already verified as valid, is discovered to be referring to a nonexistent disk. *System inconsistency, irrecoverable.*

panic: iinit

The super-block of the root file system could not be read. This message occurs only at load time. *Hardware, irrecoverable.*

panic: IO err in swap

A fatal I/O error occurred while reading or writing the swap area. *Hardware, irrecoverable.*

panic: memory failure - parity error

A hardware memory failure trap has been taken. *System inconsistency, irrecoverable.*

panic: memory management failure

An error occurred during memory management operations. *System inconsistency, irrecoverable.*

panic: no fs

A file system descriptor has disappeared from its table. *System inconsistency, irrecoverable.*

panic: no imt

A mounted file system has disappeared from the mount table. *System inconsistency, irrecoverable.*

panic: no procs

You are limited in the amount of simultaneous processes you can have; an attempt to create a new process when none is available or when your limit is exceeded is refused. That is an occasional event and produces no console messages; this panic occurs when the kernel has certified that a free process table entry is available and yet can't find one when it goes to get it. *System inconsistency, irrecoverable.*

panic out of swap

There is insufficient space on the swap disk to hold a task. The system refuses to create tasks when it feels there is insufficient disk space, but it is possible to create situations to fool this mechanism. *Abnormal, irrecoverable.*

panic: general protection trap

General protection trap taken in kernel. *System inconsistency, irrecoverable.*

panic: segment not present

An attempt has been made to access an invalid segment. It may also indicate the segment-not-present trap has been taken in the kernel. *System inconsistency, irrecoverable.*

panic: timeout table overflow

The timeout table is full. Timeout requests are generated by device drivers; there should usually be room for one entry per system serial line plus ten more for other usages. *System inconsistency, irrecoverable.*

panic: trap in system

The CPU has generated an illegal instruction trap while executing kernel or device driver code. This message is preceded by an information dump describing the trap. *System inconsistency, irrecoverable.*

panic: invalid TSS

Internal tables have become corrupted. *System inconsistency, irrecoverable.*

panic: unknown interrupt

The system received an interrupt via a supposedly unused vector. Typically, this event happens when a hardware failure miscomputes the vector of a valid interrupt. *Hardware, irrecoverable.*

proc on q

The system attempts to queue a process already on the process ready-to-run queue. *System inconsistency, irrecoverable.*

saint: received interrupt at wrong level (*num*)

This message indicates a serious hardware malfunction in the system serial devices. *Num* indicates the specific device. *Hardware.*

spurious hb interrupt

This message indicates a serious hardware malfunction associated with the fixed disk. *Hardware.*

spurious kb interrupt

This message indicates a serious hardware malfunction associated with the system console. *Hardware.*

timed out on a diskette drive *num*

An attempt to read from or write to a diskette drive has failed. The drive door may be open, the diskette in the drive may be bad, or the disk may have the wrong density for the requested read or write. *Hardware.*

trap *type*

This message precedes a "panic" message. The *type* is the trap number given by the processor. The message is followed by a dump of registers. *System inconsistency, irrecoverable.*

Warning: invalid partition table

The partition table on the fixed disk drive is not valid. The drive should not be used until **fdisk(M)** has been run to properly initialize the disk. *Hardware.*

See Also

config(CP)

MICNET(M)

Name

Micnet - The Micnet default commands file.

Description

The **Micnet** file lists the system commands that may be executed through the **remote** command. The file is required for each system in a Micnet network. Whenever a **remote** command is received through the network, the Micnet programs search the **Micnet** file for the system command specified with the **remote** command. If found, the command is executed. Otherwise, the command is ignored and an error message is returned to the system that issued the **remote** command.

The file may contain one or more lines. If all commands may be executed, then only the line:

```
executeall
```

is required in the file. Otherwise, the commands must be listed individually. A line that defines an individual command has the form:

command=commandpath

Command is the command name to be specified in a **remote** command. *Commandpath* is the full path name of the command on the specified system. The equal sign (=) separates the command and commandpath. For example, the line:

```
cat=/bin/cat
```

defines the command name **cat** (used in the **remote** command) to refer to the system command **cat** in the **/bin** directory.

When executeall is set, commands are sought in a series of default directories. Initially, the directories are **/bin** and **/usr/bin**. The default directories can be explicitly defined in the file by including a line of the form:

execpath=PATH=*directory*[:*directory*] . . .

The first part of the line, execpath=PATH=, is required. Each *directory* must be a valid pathname. The colon is required to separate directories. For example, the line:

```
execpath=PATH=/bin:/usr/bin:/usr/bobf/bin
```

sets the default directories to **/bin**, **/usr/bin**, and **/usr/bobf/bin**.

Files

/etc/default/micnet

See Also

aliases(M), netutil(C), systemid(M), top(M)

Comment

The **rcp** command cannot be executed from a remote system unless the **Micnet** file contains either executeall or the line:

```
rcp=/usr/bin/rcp
```

NULL(M)

Name

null - The null file.

Description

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

Files

/dev/null

PASSWD(M)

Name

passwd - The password file.

Description

The **passwd** file contains the following information for each user:

-Login name

-Encrypted password

-Numerical user ID

-Numerical group ID

-Comment

-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 comment can contain any desired information. Each user is separated from the next by a newline. If the password field is null, no password is demanded; if the shell field is null, **sh**(C) is used.

This file resides in the directory **/etc.** Because the passwords are encrypted, the file has general read permission and can be used, for example, to map numerical user IDs to names.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (**.**, **/**, **0-9**, **A-Z**, **a-z**) except when the password is null, in which case the encrypted password is also null. Password aging is in effect for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced by the super-user). The first character of the age denotes the maximum number of weeks for which a password is valid. A user who attempts to log in after his password has expired is forced to supply a new one. The next character denotes the minimum period in weeks that must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) The first and second characters must have numerical values in the range 0-63, where the dot (.) is equal to 0 and lowercase z is equal to 63. If the numerical value of both characters is 0, the user will be forced to change his password the next time he logs in. If the second character is greater than the first, only the super-user will be able to change the password.

Files

/etc/passwd

See Also

login(M), passwd(C), group(M), pwadmin(C)

PROFILE(M)

Name

profile - Sets up an environment at login time.

Description

The optional file **.profile** permits automatic execution of commands whenever a user logs in. The file is generally used to personalize a user's work environment by setting exported environment variables and terminal mode (see **environ (C)**).

When a user logs in, the user's login shell looks for **.profile** in the login directory. If it is found, the shell executes the commands in the file before beginning the session. The commands in the file must have the same format as if typed at the keyboard. Any line beginning with the number sign (#) is considered a comment and is ignored. The following is an example of a typical file:

```
# Tell me when new mail comes in
MAIL=/usr/mail/myname
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
```

Note that the file **/etc/profile** is a system-wide profile that, if it exists, is executed for every user before the user's **.profile** is executed.

Files

$HOME/.profile
/etc/profile

See Also

env(C), login(M), mail(C), sh(C), stty(C), su(C), environ(M)

SERIAL(M)

Name

tty**00**, tty**01** - Interface to serial ports.

Description

The **tty00** and **tty01** files provide access to the optional serial ports of the IBM Personal Computer AT. Each file corresponds to one serial port and may only be accessed if the corresponding serial interface card has been installed and its jumper address correctly set. The file **tty00** (minor number **0**) provides access to the serial port at jumper address "x03f8"; **tty01** (minor number **1**) provides access to the port at jumper address "x02f8". The serial ports must also be defined in the system configuration database (see **cmos**(M)). It is an error to attempt to access a serial port that has not been installed and defined.

The IBM Personal Computer AT serial ports can be used for a variety of serial communication purposes such as connecting login terminals to the computer or forming a serial network with other computers. Note that a serial port may operate at most of the standard XENIX baud rates, and that the ports have a DTE (data terminal equipment) configuration. The following table defines how each pin is used.

Pin	**Description**
2	Transmit Data
3	Receive Data
6	Request to Send
7	Signal Ground
8	Carrier Detect (Data Set Ready)
20	Data Terminal Ready

See **tty**(M) for the details of serial line operation in the XENIX system.

Files

/dev/tty**00**/
/dev/tty**01**

See Also

cmos(M), getty(M), tty(M)

SETCLOCK(M)

Name

setclock - Sets the real-time clock.

Syntax

setclock [*time*]

Description

The **setclock** command sets the battery-powered, real-time clock of the IBM Personal Computer AT to the given *time*. If *time* is not given, the current contents of the battery-powered clock are displayed. The *time* must be a combination of digits with the form:

```
MMddhhmm[yy]
```

where *MM* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *yy* is the last two digits of the year. If *yy* is not given, it is taken from the current system time. For example, the command:

```
setclock 0122151884
```

sets the real-time clock to 15:18 on January 22, 1984.

File

/etc/setclock

See Also

clock(M)

SETKEY(M)

Name

setkey - Assigns the function keys.

Syntax

setkey *keynum string*

Description

The **setkey** command assigns the given *string* to be the output of the computer function key given by *keynum*. For example, the command:

```
setkey 1 date
```

assigns the string "date" as the output of function key 1. The *string* can contain control characters, such as a newline character, and should be quoted to protect it from processing by the shell. For example, the command:

```
setkey 2 "pwd ; lc"
```

assigns the command sequence "pwd ; lc" to function key 2. Notice how the newline character is embedded in the quoted string. This causes the commands to be carried out when function key 2 is pressed. Otherwise, the Enter key would have to be pressed after pressing the function key.

Files

/bin/setkey

SYSTEMID(M)

Name

systemid - The Micnet system identification file.

Description

The **systemid** file contains the machine and site names for a system in a Micnet network. A *machine name* identifies a system and distinguishes it from other systems in the same network. A *site name* identifies the network to which a system belongs and distinguishes the network from other networks in the same chain.

The **systemid** file may contain a *site name* and up to four different *machine names*. The file has the form:

```
[site-name]
machine-name1
[machine-name2]
[machine-name3]
[machine-name4]
```

The file must contain at least one machine name. The other machine names are optional, serving as alternate names for the same machine. The file must contain a site name if more than one machine name is given or if the network is connected to another through a uucp link. The site name, when given, must be on the first line.

Each name can have up to eight letters and numbers but must always begin with a letter. There is never more than one name to a line. A line beginning with a number sign (#) is considered a comment line and is ignored.

The Micnet network requires one **systemid** file on each system in a network, with each file containing a unique set of machine names. If the network is connected to another network through a uucp link, each file in the network must contain the same site name.

The **systemid** file is used primarily during resolution of aliases. When aliases contain site and/or machine names, the name is compared with the names in the file and removed if there is a match. If there is no match, the alias (and associated message, file, or command) is passed to the specified site or machine for further processing.

Files

/etc/systemid

See Also

aliases(**M**), netutil(C), top(**M**)

Name

term - Conventional names.

Description

These names are maintained as part of the shell environment (see **sh**(C), **profile**(M), and **environ**(M)) in the variable $TERM:

1520	Datamedia 1520
1620	Diablo®[1] 1620 and others using the HyType II printer
1620-12	same, in 12-pitch mode
2621	Hewlett-Packard HP®[2] 2621 series
2631	Hewlett-Packard 2631 line printer
2631-c	Hewlett-Packard 2631 line printer - compressed mode
2631-e	Hewlett-Packard 2631 line printer - expanded mode
2640	Hewlett-Packard HP 2640 series
2645	Hewlett-Packard HP 264n series (other than the 2640 series)
300	DASI/DTC/GSI 300 and others using the HyType I printer
300-12	same, in 12-pitch mode
300s	DASI/DTC/GSI 300s
382	DTC 382
300s-12	same, in 12-pitch mode
3045	Datamedia 3045
33	TELETYPE®[3] Model 33 KSR
37	TELETYPE Model 37 KSR
40-2	TELETYPE Model 40/2
4000A	Trendata4000A
4014	Tektronix®[4] 4014
43	TELETYPE Model 43 KSR
450	DASI 450 (same as Diablo 1620)
450-12	same, in 12-pitch mode

[1] Diablo is a registered trademark of the Xerox Corporation.
[2] HP is a registered trademark of the Hewlett-Packard Corporation.
[3] Teletype is a registered trademark of the Teletype Corporation.
[4] Tektronix is a registered trademark of Tektronix, Inc.

735	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
tn1200	General Electric TermiNet 1200
tn300	General Electric TermiNet 300

Up to 8 characters, chosen from [**-**, **a-z**, **0-9**] make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a **-**. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept arguments of the form **- T***term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **$TERM**, which, in turn, should contain *term*.

See Also

mm(CT), nroff(CT), sh(C), stty(C), profile(M), environ(M).

Comment

Not all XENIX facilities support all of these options.

TERMCAP(M)

Name

termcap - Terminal capability data base.

Description

The file **/etc/termcap** is a data base describing terminals. This data base is used by programs such as **vi**(C). Terminals are described in **termcap** by giving a set of capabilities and by describing how operations are performed. Padding requirements and initialization sequences are included in **termcap**.

Entries in **termcap** consist of a number of ':' separated fields. The first entry for each terminal gives the names that are known for the terminal, separated by vertical bar (|) characters. The first name is always two characters long for compatibility with older level systems. 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 contain blanks for readability.

Capabilities

The following is a list of the capabilities that can be defined for a given terminal. In this list, (P) indicates that padding may be specified, (P*) indicates that padding may be based on the number of lines affected, and uppercase names indicate XENIX extensions (except for CC).

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**
BE	str		Bell character
bs	bool		Terminal can backspace with ∧**H**
BS	str		Sent by Bksp key (if not bc)
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
CF	str		Cursor off
ch	str	(P)	Like cm but horizontal motion only, line stays same
CL	str		Sent by CHAR LEFT key
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
CN	str		Sent by CANCEL key
co	num		Number of columns in a line
CO	str		Cursor on
CR	str		Sent by CHAR RIGHT key
cr	str	(P*)	Carriage return, (default ∧**M)**
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
CW	str		Sent by CHANGE WINDOW key
da	bool		Display may be retained above
db	bool		Display may be retained below

Name	Type	Pad	Description
dB	num		Number of millisec of bs delay needed
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
DK	str		Sent by down arrow key (if not kd)
DL	str		Sent by Del key
DL	str		Sent by destructive character delete key
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
EE	str		Edit mode end
EG	num		Number of chars taken by ES and EE
ei	str		End insert mode; give ':ei=:' if **ic**
EN	str		Sent by End key
eo	str		Can erase overstrikes with a blank
ES	str		Edit mode start
ff	str	(P*)	Hardcopy terminal page eject (default ^**L**)
G1	str		Upper-right (1st quadrant) corner character
G2	str		Upper-left (2nd quadrant) corner character
G3	str		Lower-left (3rd quadrant) corner character
G4	str		Lower-right (4th quadrant) corner character
GD	str		Down-tick character
GE	str		Graphics mode end
GG	num		Number of chars taken by GS and GE
GH	str		Horizontal bar character
GS	str		Graphics mode start
GU	str		Up-tick character

Name	Type	Pad	Description
GV	str		Vertical bar character
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
HM	str		Sent by Home key (if not kh)
ho	str		Home cursor (if no **m**)
HP	str		Sent by HELP key
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=:q' if**ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by 'other' function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of 'keypad transmit' mode
KF	str		Key-click off
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of 'other' keys
KO	str		Key-click on
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in 'keypad transmit' mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on 'other' function keys
LD	str		Sent by line delete key
LF	str		Sent by line feed key
li	num		Number of lines on screen or page

Name	Type	Pad	Description
LK	str		Sent by left arrow key (if not kl)
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
MN	str		Sent by minus sign key
MP	str		Multiplan initialization string
MR	str		Multiplan reset string
mu	str		Memory unlock (turn off memory lock)
nc	bool		No correctly working carriage return (DM2500, H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll
NU	str		Sent by NEXT UNLOCKED CELL key
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
PD	str		Sent by PAGE DOWN key
PL	str		Sent by PAGE LEFT key
PR	str		Sent by PAGE RIGHT key
PS	str		Sent by plus sign key
pt	bool		Has hardware tabs (may need to be set with **is**)
PU	str		Sent by PAGE UP key
RC	str		Sent by RECALC key
RF	str		Sent by TOGGLE REFERENCE key
RK	str		Sent by right arrow key (if not kr)
RT	str		Sent by RETURN key
RT	str		Sent by return key
se	str		End stand out mode
sf	str	(P)	Scroll forward

Name	Type	Pad	Description
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)
TB	str		Sent by Tab key
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
UK	str		Sent by up arrow key (if not ku)
ul	bool		Terminal underlines even though it doesn't 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
WL	str		Sent by WORD LEFT key
WR	str		Sent by WORD RIGHT key
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like **ce**\r\n (Delta Data)
xs	bool		Standard out not erased by writing over it (HP 2641)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry describes the Concept-100, and is among the more complex entries in the **termcap** file. (This particular concept entry is used as an example only.):

```
c1|c100 |concept100:is=\EU\Ef\E7\E5\
E8\El\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:\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 empty fields may be included for readability. Capabilities in **termcap** are of three types: Boolean capabilities, which indicate that the terminal has some particular feature, numeric capabilities give the size of the terminal or the size of particular delays, and string capabilities, that give a sequence that can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has automatic margins (that is, an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Numeric capabilities are followed by the character # and 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, for example 20, or an integer followed by an *, that is, 3*. A * indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a * is specified, it is sometimes useful to give a delay of the form 3.5 to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, ^**x** maps to a control-x for any appropriate **x**, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace, and formfeed. Finally, characters may be given as three octal digits after a \ , and the characters ^ and \ may be given as \ ^ and \ \ . If it is necessary to place a **:** in a capability it must be escaped in octal as **\072** . If it is necessary to place a null character in a string capability it must be encoded as **\200** . The routines that deal with **termcap** use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

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.

Basic Capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, it should have the **am** capability. If the terminal can clear its screen, this is given by the **cl** string capability. If the terminal can backspace, it should have the **bs** capability, unless a backspace is accomplished by a character other than ∧**H** in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) 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 never attempts to backspace around the left edge, nor will it attempt to go up off the top. The editor assumes that feeding off the bottom of the screen causes 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, that is, **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 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, 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 0140 (DM2500).
%**B**	BCD (16*(x/10)) + (x%10), no output.
%**D**	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c$_7$.pslzero2 3Y padded for 6 milliseconds. The order of the rows and columns is inverted here, and the row and column are printed as two digits. Its **cm** capability is 'cm=6\E&%r%2c%2Y'. The Microterm ACT-IV needs the current row and column sent preceded by a ∧**T** , with the row and column simply encoded in binary, 'cm=∧T%.%.'. Terminals that use '%.' need to be able to backspace the cursor (**bs** or **bc**) and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t** , **\n** , ∧**D** and **\r** , as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus 'cm=\E=%+ %+'.

Cursor Motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, 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), 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, 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 that the cursor is on, 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 backward, 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 nonblank lines up from below or that scrolling back with **sb** may bring down nonblank 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. 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, 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, you may have to modify the editor to get it to use the insert mode your terminal defines. No known terminals have an insert mode not falling into one of these two classes.

The editor can handle both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. 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** an empty value). 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 that send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, 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 standout modes (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 1061 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) 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, for example, from an underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by 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. 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=** ^**K**j^**Z**k^**Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than ^**I** to tab, 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. Because **termlib** routines search the entry from left to right, and because 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 cancelled with **xx@** where xx is the capability. For example:

```
hn|2621nl:ks@:ke@:tc=2621:
```

This defines a 2621nl that does not have the **ks** or **ke** capabilities, and 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(C), tset(C), vi(C), more(C)

Credit

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

Comments

The **ex** editor allows only 256 characters for string capabilities. 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.

TERMINALS(M)

Name

terminals - List of supported terminals.

Description

The following is a partial list of terminals that may be used on a XENIX system. The corresponding *names* can be used to assign the terminal type to TERM (see **environ**(M)). Additional terminals and names are defined in the file:

/etc/termcap.

Name	**Terminal**
dw1	DECwriter I
dw2	DECwriter II
h19	Heathkit h19
h1ØØØ	Hazeltine 1ØØØ
h1552	Hazeltine 1552
h15ØØ	Hazeltine 15ØØ
h151Ø	Hazeltine 151Ø
h152Ø	Hazeltine 152Ø
h2ØØØ	Hazeltine 2ØØØ
3101	IBM 3101
ti7ØØ	Texas Instruments Silent 7ØØ
ti745	Texas Instruments Silent 745
vt5Ø	Digital VT5Ø
vt5Øh	Digital VT5Øh
vt52	Digital VT52

Name	Terminal
vt1	Digital VT1ØØ
vt1ØØs	Digital VT1ØØ 132 cols 14 lines
vt1ØØw	Digital VT1ØØ 132 cols
z19	Zenith Z19
z29	Zenith Z29

Files

/etc/termcap

Name

top, top.next - The Micnet topology files.

Description

These files contain the topology information for a Micnet network. The topology information describes how the individual systems in the network are connected and what path a message must take from one system to reach another. Each file contains one or more lines of text. Each line of text defines a connection or a communication path.

The **top** file defines connections between systems. Each line lists the machine names of the connected systems, the serial lines used to make the connection, and the speed (baud rate) of transmission between the systems. Each line has the form:

machine1 tty1 machine2 tty2 speed

where *machine1* and *machine2* are the machine names of the respective systems (as given in the **systemid** files), and *tty1* and *tty2* are the device names (for example, tty01) of the connecting serial lines. The speed must be an acceptable baud rate (for example, 110, 300, . . . , 2400).

The **top.next** file contains information about how to reach a particular system from a given system. There may be several lines for each system in the network. Each line lists the machine name of a system, followed by the machine name of a system connected to it, followed by the machine names of all the systems that may be reached by going through the second system. Such a line has the form:

machine1 machine2 machine3 [*machine4*] . . .

The machine names must be the names of the respective systems (as given by the first machine name in the **systemid** files).

The **top.next** file must be present even if there are only two computers in the network. In such a case, the file must be empty.

In the **top** and **top.next** files, any line beginning with a number sign (#) is considered a comment and is ignored.

Files

/usr/lib/mail/top
/usr/lib/mail/top.next

See Also

aliases(M), netutil(C), systemid(M), top(M)

TTY(M)

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 the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

All asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by **getty**(M) and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the control terminal for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below.

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 full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. This limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been 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.

Erase and kill processing is normally done during input. By default, pressing Ctrl-H or Backspace erases the last character typed, except that it will not erase beyond the beginning of the line. By default, Ctrl-U deletes the entire input line, and optionally outputs a newline character. Both these characters operate on a key-stroke basis, independent of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\) . In this case the escape character is not read. The erase and kill characters may be changed (see **stty**(C)).

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII Del) Generates an *interrupt* signal that 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.

QUIT (Ctrl-\ or ASCII FS) Generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

ERASE (Ctrl-H) Erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (Ctrl-U) Deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Ctrl-D or ASCII EOT) May be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters are passed back, which is the standard end-of-file indication.

NL (ASCII LF) Is the normal line delimiter. It cannot be changed or escaped.

EOL (ASCII Nul) Is an additional line delimiter, like NL. It is not normally used.

STOP (Ctrl-S or ASCII DC3) Can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Ctrl-Q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters cannot be changed or escaped.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL can be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is carried out.

When the carrier signal from the dataset drops, a *hangup* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it is suspended when its output queue exceeds a given limit. When the queue has drained down to the given threshold, the program is resumed.

Several system calls apply to terminal files. The primary calls use the following structure, defined in the file **termio.h**:

```
#define NCC          8
struct  termio {
        unsigned     short   c_iflag;      /* input modes */
        unsigned     short   c_oflag;      /* output modes */
        unsigned     short   c_cflag;      /* control modes */
        unsigned     short   c_lflag;      /* local modes */
        char                 c_line;       /* line discipline */
        unsigned     char    c_cc[NCC];    /* control chars */
};
```

The special control characters are defined by the array *c__cc*. The relative positions and initial values for each function are as follows:

0	INTR	Del
1	QUIT	FS
2	ERASE	Bksp
3	KILL	Ctrl-U, Ctrl-H,
4	EOF	EOT
5	EOL	Nul
6	Reserved	
7	Reserved	

The *c__iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignores break condition
BRKINT	0000002	Signals interrupt on break
IGNPAR	0000004	Ignores characters with parity errors
PARMRK	0000010	Marks parity errors
INPCK	0000020	Enables input parity check
ISTRIP	0000040	Strips character
INLCR	0000100	Maps NL to CR on input
IGNCR	0000200	Ignores CR
ICRNL	0000400	Maps CR to NL on input
IUCLC	0001000	Maps uppercase to lowercase on input
IXON	0002000	Enables start/stop output control
IXANY	0004000	Enables any character to restart output
IXOFF	0010000	Enables start/stop input control

If IGNBRK is set, the break condition (a character-framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise, if BRKINT is set, the break condition generates an interrupt signal and flushes both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error, that is not ignored, is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error, that is not ignored, is read as the character Nul (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to seven-bits, otherwise all eight-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. All start/stop characters are ignored and not read. If IXANY is set, any input character restarts output which has been suspended.

If IXOFF is set, the system transmits START characters when the input queue is nearly empty and STOP characters when nearly full.

The initial input control value is all bits clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocesses output.
OLCUC	0000002	Maps lowercase to uppercase on output.
ONLCR	0000004	Maps NL to CR-NL on output.
OCRNL	0000010	Maps CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Uses fill characters for delay.
OFDEL	0000200	Fills is Del, else Nul.
NLDLY	0000400	Selects newline delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Selects carriage return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	

TABDLY	0014000	Selects horizontal tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expands tabs to spaces.
BSDLY	0020000	Selects backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Selects vertical tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Selects form feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to perform the carriage return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise the NL character is assumed to perform the linefeed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all, cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If OFDEL is set, the fill character is Del, otherwise Nul.

If a form feed or vertical tab delay is specified, it lasts for about two seconds. If OFILL is set no fill characters are transmitted.

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage return delays are used instead of the newline delays. If OFILL is set, two fill characters are transmitted.

Carriage return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, CR1,CR2, and CR3 transmits two fill characters,

Horizontal tab delay type 1 depends on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, TAB1 transmits two fill characters and TAB2 transmits one fill character.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c__cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up.
B50	0000001	50 baud.
B75	0000002	75 baud.
B110	0000003	110 baud.
B134	0000004	134.5 baud.
B150	0000005	150 baud.
B200	0000006	200 baud.
B300	0000007	300 baud.
B600	0000010	600 baud.
B1200	0000011	1200 baud.
B1800	0000012	1800 baud.
B2400	0000013	2400 baud.
B4800	0000014	4800 baud.
B9600	0000015	9600 baud.
EXTA	0000016	External A.
EXTB	0000017	External B.
CSIZE	0000060	Character size:
CS5	0	five bits.
CS6	0000020	six bits.
CS7	0000040	seven bits.
CS8	0000060	eight bits.
CSTOPB	0000100	Sends two stop bits, else one.
CREAD	0000200	Enables receiver.
PARENB		0000400 Parity enable.
PARODD		0001000 Odd parity, else even.
HUPCL	0002000	Hangs up on last close.
CLOCAL		0004000 Local line, else dial-up.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Without this signal, the line is disconnected if connected through a modem. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise 1 stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line is disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal is not asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. The data-terminal-ready and request-to-send signals are asserted, but incoming modem signals are ignored. If CLOCAL is not set, modem control is assumed. This means the data-terminal-ready and request-to-send signals are asserted. Also, the carrier-detect signal must be returned before communications can proceed.

The initial hardware control value after open is B9600, CS8, CREAD, HUPCL.

The *c__lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enables echo.
ECHOE	0000020	Echoes erase character as BS-SP-BS.
ECHOK	0000040	Echoes NL after kill character.
ECHONL	0000100	Echoes NL.
NOFLSH	0000200	Disables flush after interrupt or quit.
XCLUDE	0100000	Exclusive use of the line.

p. If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus, these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (for example, 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least VMIN characters have been received or the timeout value VTIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The VMIN and VTIME values are stored in the position for the EOF and EOL characters respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an uppercase letter is accepted on input by preceding it with a \ character and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

For:	**Use:**
\`	\\'
|	\\!
~	\\^
{	\\(
}	\\)
\\	\\\\

For example, **A** is input as \a , \n as \\n , and \N as \\\n .

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which clears the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character is echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character is echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters, is not done.

If XCLUDE is set, any subsequent attempt to open the tty device fails for all users except the super-user. If the call fails, it returns EBUSY in *errno.* XCLUDE is useful for programs that must have exclusive use of a communications line. It is not intended for the line to the program's controlling terminal. XCLUDE must be cleared before the setting program terminates, otherwise subsequent attempts to open the device will fail.

The initial line-discipline control value is all bits clear.

The primary **ioctl** system calls have the form:

```
ioctl (fildes, command, arg)
struct termio arg;
```

The commands using this form are:

TCGETA Gets the parameters associated with the terminal and stores them in the *termio* structure referenced by **arg**.

TCSETA Sets the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW Waits for the output to drain before setting the new parameters. This form should be used when changing parameters that affect output.

TCSETAF Waits for the output to drain, then flushes the input queue and sets the new parameters.

Additional **ioctl** calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK	Waits for the output to drain. If *arg* is 0, sends a break (zero bits for 0.25 seconds).
TCXONC	Starts/stops control. If *arg* is 0, suspends output; if 1, restarts suspended output.
TCFLSH	If *arg* is 0, flushes the input queue; if 1, flushes the output queue; if 2, flushes both the input and output queues.

Files

/dev/tty
/dev/tty*
/dev/console

See Also

stty(C)

TTYS(M)

Name

ttys - Log in terminals file.

Description

The **/etc/ttys** file contains a list of the device special files associated with possible log in terminals and defines which files are to be opened by the **init (M)** program on system start-up.

The file contains one or more entries of the form:

state mode name

The *name* must be the filename of a device special file. Only the filename may be supplied; the path is assumed to be **/dev**. If *state* is "1", the file is enabled for logins; if "0", the file is disabled. The *mode* is used as an argument to the **getty**(M) program. It defines the line speed and type of device associated with the terminal. A list of arguments is provided in **getty**(M).

For example, the entry "16tty02" means the serial line tty02 is to be opened for logging in at 9600 baud.

Files

/etc/ttys

See Also

init(M), getty(M), enable(C), disable(C)

UTMP(M)

Name

utmp, wtmp - Formats of utmp and wtmp entries.

Description

The files **utmp** and **wtmp** hold user and accounting information for use by commands such as **who**(C), and **login**(M). They have the following structure, as defined by **/usr/include/utmp.h**:

```
struct utmp
{
        char    ut_line[8];     /* tty name */
        char    ut_name[8];     /* login name */
        long    ut_time;        /* time on */
};
```

Files

/etc/utmp
/usr/adm/wtmp
/usr/include/utmp.h

See Also

login(M), who(C), write(C)

XINSTALL(M)

Name

xinstall - Installs XENIX systems.

Syntax

xinstall [*option*]

Description

The **xinstall** command copies the XENIX program and data files from the XENIX system distribution disks to the root and user file systems of a fixed disk. The **xinstall** command creates the appropriate directories for the files and sets permissions. Existing files of the same name are deleted.

The *options* may be any one of the following:

base	Install the XENIX Operating System.
soft	Install the XENIX Software Development System.
text	Install the XENIX Text Formatting System.

If no option is given, **base** is assumed.

The **xinstall** command first performs a five or six step check of the file system, then explains how to insert the distribution disks into the system's diskette drive. It displays the message:

```
First Diskette? [y,n]
```

when it is ready for the first disk. It displays a similar message when ready for the next disk. All distribution disks are numbered and must be inserted in order.

In some cases, programs or data files may be split across two disks. When a split file is encountered, **xinstall** displays the message:

```
tar:  please mount new volume
```

to indicate that the next disk in the set must be inserted.

When all disks have been copied, **xinstall** assigns permissions for each file using the **/etc/fixperm** program.

Files

/etc/xinstall
/etc/fixperm

See Also

XENIX Installation Guide

Section 3. File Formats

Introduction to File Formats(F)

Name

intro - Introduction to file formats.

Description

This section outlines the formats of various files. Usually, these structures can be found in the directories **/usr/include** or **/usr/include/sys**.

A.OUT(F)

Name

a.out - Format of assembler and link editor output.

Description

The **a.out** file is the output file of the assembler **as** and the link editor **ld**. Both programs make **a.out** executable if there were no errors in assembling or linking, and no unresolved external references.

The format of **a.out**, called the **x.out** or segmented **x.out** format, is defined by the files **/usr/include/a.out.h** and **/usr/include/sys/relsym.h**. The **a.out** file has the following general layout:

1. Header.

2. Extended header.

3. File segment table (for segmented formats).

4. Segments (Text, Data, Symbol, and Relocation).

In the segmented format, there may be several text and data segments, depending on the memory model of the program. Segments within the file begin on boundaries that are multiplies of 512 bytes as defined by the file's pagesize.

ACCT(F)

Name

acct - Format of per-process accounting file.

Description

Files produced as a result of calling **acct** have records in the form defined by **<sys/acct.h>**.

In *ac_flag*, the AFORK flag is turned on by each fork and turned off by each *exec*. The *ac_comm* field is inherited from the parent process and is reset by any *exec*. Each time the system charges the process with a clock tick, it also adds the current process size to *ac_mem* computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of *ac_mem/ac_stime* can be viewed as an approximation to the mean process size, as modified by text-sharing.

See Also

acct(C), acctcom(C)

Comment

The *ac_mem* value for a short-lived command gives little information about the actual size of the command, because *ac_mem* may be incremented while a different command (for example, the shell) is being executed by the process.

AR(F)

Name

ar - Archive file format.

Description

The archive command **ar** is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor **ld**(M).

A file produced by **ar** has a magic number at the start, followed by the constituent files; each preceded by a file header. The magic number is Ø177545 octal (or 0xff65 hexadecimal). The header of each file is declared in **/usr/include/ar.h**.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless, the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

BACKUP(F)

Name

backup - Incremental dump tape format.

Description

The **backup** and **restore** commands are used to write and read incremental dump magnetic tapes.

The backup tape consists of a header record, some bit mask records, a group of records describing file system directories, a group of records describing file system files, and some records describing a second bit mask.

The header record and the first record of each description have the format described by the structure included by:

#include <dumprestor.h>

Fields in the **dumprestor** structure are described below.

NTREC is the number of 512 byte blocks in a physical tape record. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are:

TS_TYPE	Tape volume label.
TS_INODE	A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.
TS_BITS	A bit mask follows. This bit mask has a 1 bit for each inode that was dumped.
TS_ADDR	A subblock to a file (TS_INODE). See the description of *c_count* below.

TS__END	End of tape record.
TS__CLRI	A bit mask follows. This bit mask contains a 1 bit for all inodes that were empty on the file system when dumped.
MAGIC	All header blocks have this number in *c__magic* .
CHECKSUM	Header blocks checksum to this value.

The fields of the header structure are:

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 block number of this record. This is counting 512 byte blocks.
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 block sum to CHECKSUM.
c__dinode	This is a copy of the inode as it appears on the file system.

c_count	This is the count of characters following that describe the file. A character is 0 if the block associated with that character was not present on the file system; otherwise the character is nonzero. If the block was not present on the file system, no block was dumped and it is replaced as a hole in the file. If there is not sufficient space in this block to describe all of the blocks in a file, TS_ADDR blocks are scattered through the file, each one picking up where the last left off.
c_addr	This is the array of characters that is used as described above.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END block and a tapemark.

The structure **idates** describes an entry of the file where dump history is kept.

See Also

backup(C), restore(C), filesystem(F)

CHECKLIST(F)

Name

checklist - List of file systems processed by **fsck**.

Description

The **/etc/checklist** file contains a list of the file systems to be checked when **fsck**(C) is invoked without arguments. The list contains, at most, 15 *special file* names. Each *special file* name must be on a separate line and must correspond to a file system.

See Also

fsck(C)

CORE(F)

Name

core - Format of core image file.

Description

XENIX writes out a **core** image of a terminated process when various errors occur. The most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called **core** and is written in the process' working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID does not produce a core image.

The first section of the **core** image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter *usize*, which is defined in **/usr/include/sys/param.h.** The remainder represents the actual contents of the user's **core** area when the **core** image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in **/usr/include/sys/user.h**. The locations of registers, are outlined in **/usr/include/sys/reg.h**.

CPIO(F)

Name

cpio - Format of cpio archive.

Description

The *header* structure, when the **c** option is not used, is:

```
struct {
        short   h_magic,
                h_dev,
                h_ino,
                h_mode,
                h_uid,
                h_gid,
                h_nlink,
                h_rdev,
                h_mtime[2],
                h_namesize,
                h_filesize[2];
        char    h_name[h_namesize rounded to word];
} Hdr;
```

When the **c** option is used, the *header* information is described by the statement below:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11o%6o%6o%s",
       &Hdr.h_magic,&Hdr.h_dev,&Hdr.h_ino,&Hdr.h_mode,
       &Hdr.h_uid,&Hdr.h_gid,&Hdr.h_nlink,&Hdr.h_rdev
       &Longtime,&Hdr.h_namesize,&Longfile,Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h__mtime* and *Hdr.h__filesize*, respectively. The contents of each file is recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h__magic* contains the constant 070707 (octal). The length of the null-terminated pathname *h__name*, including the null byte, is given by *h__namesize*.

The last record of the *archive* always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with *h__filesize* equal to zero.

See Also

cpio(C), find(C)

DIR(F)

Name

dir - Format of a directory.

Syntax

```
#include <sys/dir.h>
```

Description

A directory behaves exactly like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry (see **filesystem**(F)). The structure of a directory is given in the include file **/usr/include/sys/dir.h**.

By convention, the first two entries in each directory are "dot" (.) and "dot dot" (..). The first is an entry for the directory itself. The second is for the parent directory. The meaning of dot dot is modified for the root directory of the master file system; there is no parent, so dot dot has the same meaning as dot.

See Also

filesystem(F)

FILESYSTEM(F)

Name

file system - Format of a system volume.

Syntax

```
#include <sys/filsys.h>
#include <sys/types.h>
#include <sys/param.h>
```

Description

Every file system storage volume (for example, a fixed disk) has a common format for certain vital information. Every such volume is divided into a certain number of 512 word (1,024 byte) blocks. Block 0 is unused and is available to contain a bootstrap program or other information.

Block 1 is the *super-block*. The format of a super-block is described in **/usr/include/sys/filesys.h**. In that include file, *S__isize* is the address of the first data block after the i-list. The i-list starts just after the super-block in block 2; thus the i-list is *s__isize*- 2 blocks long. *S__fsize* is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers. If an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the console. Moreover, the free array is cleared 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 49 numbers of free blocks. *S__free*[0] is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 longs of this chain member. The first of these 50 blocks is the link to the next member of the chain.

To allocate a block: decrement *s_nfree*, and the new block is *s_free*[*s_nfree*]. If the new block number is 0, there are no blocks left, so give an error. If *s_nfree* becomes 0, read in the block named by the new block number, replace *s_nfree* by its first word, and copy the block numbers in the next 50 longs into the *s_free* array. To free a block, check if *s_nfree* is 50; if so, copy *s_nfree* and the *s_free* array into it, write it out, and set *s_nfree* to 0. In any event set *s_free*[*s_nfree*] to the freed block's number and increment *s_nfree*.

S_tfree is the total free blocks available in the file system.

S_ninode is the number of free i-numbers in the *s_inode* array. To allocate an inode: if *s_ninode* is greater than 0, decrement it and return *s_inode*[*s_ninode*]. If it was 0, read the i-list and place the numbers of all free inodes (up to 100) into the *s_inode* array, then try again. To free an inode, provided *s_ninode* is less than 100, place its number into *s_inode*[*s_ninode*] and increment *s_ninode*. If *s_ninode* is already 100, do not bother to enter the freed inode into any table. This list of inodes only speeds up the allocation process. The information about whether the inode is really free is maintained in the inode itself.

S_tinode is the total free inodes available in the file system.

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 also immaterial, and is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

S_ronly is a read-only flag to indicate write-protection.

S_time is the last time the super-block of the file system was changed, and is a double-precision representation of the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reload, the *s_time* of the super-block for the root file system is used to set the system's idea of the time. The system time is not set from the real time **cmos** clock when multiuser mode is entered.

I-numbers begin at 1, and the storage for inodes begins in block 2. Also, inodes are 64 bytes long, so 8 of them fit into a block. Therefore, inode *i* is located in block (i+15)/8, and begins 64×((i+15) (mod 8)) bytes from its start. Inode 1 is reserved for future use. Inode 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each inode represents one file. For the format of an inode and its flags, see **inode**(F).

Files

/usr/include/sys/filsys.h
/usr/include/sys/stat.h

See Also

fsck(C), mkfs(C), inode(F)

INODE(F)

Name

inode - Format of an inode.

Syntax

```
#include <sys/types.h>
#include <sys/ino.h>
```

Description

An inode for a plain file or directory in a file system has the structure defined by **<sys/ino.h>**. For the meaning of the defined types *off__t* and *time__t* see **types**(F).

Files

/usr/include/sys/ino.h

See Also

filesystem(F), types(F)

MASTER(F)

Name

master - master device information table

Description

This file is used to obtain device information that enables it to generate the configuration files. The file consists of four parts, each separated by a line with a dollar sign ($) in column 1. Part 1 contains device information; part 2 contains the line discipline table; part 3 contains names of devices that have aliases; part 4 contains tunable parameter information. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1

This part contains definitions for the system devices. Each line has 14 fields with the fields delimited by tabs and/or blanks:

Field 1:	device name (eight chars. maximum).
Field 2:	interrupt vector size (decimal, in bytes).
Field 3:	device mask (octal)-each "on" bit indicates that the driver has the corresponding handler or structure:

000400	not used.
000200	not used.
000100	initialization handler.
000040	not used.
000020	open handler.
000010	close handler.
000004	read handler.
000002	write handler.
000001	ioctl handler..

Field 4: device type indicator (octal):

000200	allow only one of these devices.
000100	not used.
000040	not used.
000020	required device.
000010	block device.
000004	character device.
000002	not used.
000001	not used.

Field 5:	handler prefix (4 chars. maximum).
Field 6:	not used.
Field 7:	major device number for block-type device.
Field 8:	major device number for character-type device.
Field 9:	maximum number of devices per controller (decimal).
Field 10:	not used.
Fields 11-14:	maximum of four interrupt vector addresses. Each address is followed by a unique letter or a blank.

Devices not interrupt-driven have an interrupt vector size of zero. Devices that generate interrupts but are not of the standard character or block device mold, should be specified with a type (field 4) that does not have the block or character bits set.

Part 2

This part contains definitions for the system line discipline. Each line has 11 fields. Each field is a maximum of 8 characters delimited by a blank if less than 8:

Field 1:	Device associated with this line.
Field 2:	open routine.
Field 3:	close routine.
Field 4:	read routine.
Field 5:	write routine.
Field 6:	ioctl routine.

Field 7:	receiver interrupt routine.
Field 8:	unused- should be nulldev.
Field 9:	unused- should be nulldev.
Field 10:	output start routine.
Field 11:	unused- should be nulldev.

Part 3

This part contains definitions for device aliases. Each line has 2 fields:

Field 1:	alias name of device (8 chars. maximum).
Field 2:	reference name of device (8 chars. maximum; specified in Part 1).

Part 4

This part contains the names and default values for tunable parameters. Each line has 2 or 3 fields:

Field 1:	parameter name (as it appears in description file; 20 chars. maximum).
Field 2:	parameter name (as it appears in the **c.c** file; 20 chars. maximum)
Field 3:	default parameter value (20 chars. maximum; parameter specification is required if this field is omitted)

If a parameter has no default value, an explicit specification for the parameter must be given in the description file.

Devices that are not interrupt-driven have an interrupt vector size of zero. Devices that generate interrupts but are not of the standard character or block device mold, should be specified with a type (field 4 in part 1), which has neither the block nor char bits set.

MNTTAB(F)

Name

mnttab - Format of mounted file system table.

Syntax

```
#include <stdio.h>
#include <mnttab.h>
```

Description

The **/etc/mnttab** file contains a table of devices mounted by the **mount**(C) command.

Each table entry contains the pathname of the directory on which the device is mounted, the name of the device special file, the read/write permissions of the special file, and the date on which the device was mounted.

The maximum number of entries in **mnttab** is based on the system parameter NMOUNT located in **/usr/sys/conf/c.c**, which defines the number of allowable mounted special files.

See Also

mount(C)

SCCSFILE(F)

Name

sccsfile - Format of an SCCS file.

Description

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines). Each logical part of a SCCS file is described in detail below.

Throughout an SCCS file there are lines that begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the *control character* and will be represented graphically as @. Any line described below that is not depicted as beginning with the control character is prevented from beginning with the control character. Entries of the form DDDDD represent a five digit string (a number between 00000 and 99999).

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The @hR provides a *magic number* of (octal) 064001.

Delta Table

The delta table consists of a variable number of entries of the form:

```
@s DDDDD/DDDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr>DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
@c <comments> ...
.
.
@e
```

The first line (@s) contains the number of lines inserted/deleted/unchanged respectively. The second line (@d) contains the type of the delta (currently, normal: D, and removed: R), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The @i, @x, and @g lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The @m lines (optional) each contain one MR number associated with the delta; the @c lines contain comments associated with the delta.

The @e line ends the delta table entry.

User Names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

Flags

Flags are keywords used internally. Each flag line takes the form:

```
@f <flag>      <optional text>
```

The following flags are defined:

```
@f t  <type of program>
@f v  <program name>
@f i
@f b
@f m  <module name>
@f f  <floor>
@f c  <ceiling>
@f d  <default-sid>
@f n
@f j
@f l  <lock-releases>
@f q  <user defined>
```

The t flag defines the replacement for the identification keyword. The **v** flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause an unrecoverable error (the file will not be accessed, or the delta will not be made).

When the **b** flag is present, the **-b** option may be used with the **get** command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the sccsfile.F identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a **get** command. The **n** flag causes *delta* to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes **get** to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing. The **q** flag defines the replacement for the identification keyword.

Comments

Comments are arbitrary text surrounded by the bracketing lines @t and @T. The comments section typically contains a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, as follows:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

The digit string (DDDDD) is the serial number corresponding to the delta for the control line.

STAT(F)

Name

stat - Data returned by stat system call.

Syntax

```
#include <sys/stat.h>
```

Description

The **sys/stat.h** include file contains the definition for the structure returned by the **stat** and **fstat** functions. The structure is defined as:

```
struct stat {

  dev_t      st_dev;      /* id of device containing directory
                          entry */
  ino_t      st_ino;      /* inode number */
  ushort     st_mode;     /* file mode */
  short      st_nlink;    /* # of links */
  ushort     st_uid;      /* owner uid */
  ushort     st_gid;      /* owner gid */
  dev_t      st_rdev;     /* device id (block and char
                          devices only) */
  off_t      st_size;     /* file size in bytes */
  time_t     st_atime;    /* time of last access */
  time_t     st_mtime;    /* time of last data modification */
  time_t     st_ctime;    /* time of last file
                          status 'change' */

};
```

The *st_atime*, *st_mtime*, and *st_ctime* values are measured in seconds since 00:00:00 (GMT) on January 1, 1970.

The *st_mode* value is actually a combination of one or more of the following file mode values:

S_IFMT	**0170000**	/* type of file */
S_IFDIR	**0040000**	/* directory */
S_IFCHR	**0020000**	/* character special */
S_IFBLK	**0060000**	/* block special */
S_IFREG	**0100000**	/* regular */
S_IFIFO	**0010000**	/* fifo */
S_IFNAM	**0050000**	/* name space entry */
S_INSEM	**01**	/* semaphore */
S_INSHD	**02**	/* shared memory */
S_ISUID	**04000**	/* set user id on execution */
S_ISGID	**02000**	/* set group id on execution */
S_ISVTX	**01000**	/* save swapped text even after use */
S_IREAD	**00400**	/* read permission, owner */
S_IWRITE	**00200**	/* write permission, owner */
S_IEXEC	**00100**	/* execute/search permission, owner */

Files

/usr/include/sys/stat.h

TYPES(F)

Name

types - Primitive system data types.

Syntax

#include <sys/types.h>

Description

The data **types** defined in the *include* file < **sys/types.h** > are used in XENIX system code; some data of these types are accessible to user code.

The form is as follows:

```
typedef long    daddr__t;
typedef char    *caddr__t;
typedef unsigned short  ushort;
typefde unsigned short  ino__t;
typedef char    cnt__t;
typedef long    time__t;
typedef int    label__t[6];    /* return, sp, si, di, bp*/
typedef short   dev__t;
typedef long    off__t;
typedef long    paddr__t;
typedef unsigned short      mloc__t; /* memory region location */
typedef unsigned short      msize__t;  /* memory region size */
```

```
/* selectors and constructor for device code */
#define      major(x)    (((unsigned)(x)>>8))
#define      minor(x)    ((x)&0377)
#define      makedev(x,y) (dev_t)((x)<<8|(y))

typedef unsigned char      uchar_t;
typedef unsigned long      ulong_t;
struct  saddr {
                  unsigned short  sa_seg;
                  long            sa_off;
};
```

The form *daddr_t* is used for disk addresses except in an inode on disk, see **filesystem**(F). Times are encoded in seconds since 00:00:00 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.

Files

filesystem(F)

Index

Special Characters

The Personal Computer
Programming Family

Reader's Comment Form

XENIX™ 6138656
Command Reference

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 321 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Tape Please do not staple Tape

The Personal Computer
Programming Family

Reader's Comment Form

XENIX™ 6138656
Command Reference

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 321 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

IBM

The Personal Computer
Programming Family

Reader's Comment Form

XENIX™
Command Reference

6138656

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS **PERMIT NO. 321** **BOCA RATON, FLORIDA 33432**

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

IBM

The Personal Computer
Programming Family

Reader's Comment Form

XENIX™
Command Reference

6138656

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 321 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Tape Please do not staple Tape

International Business
Machines Corporation
P.O. Box 1328-S
Boca Raton, Florida 33432

Printed in the
United States of America

6138656