AT&T

AT&T System V.3

User's Ref. Manual

089-9714-00
Version B - June 1992

Diab Data AB, Box 2029 183 02 TÄBY.
✆ + 46 8 638 94 00. Fax 08 792 05 61.

AT&T System V.3
User's Reference Manual

Flik	AT&T
1	1, 1G, 1M
2	1B
3	1C
4	6
5	References

1

NAME

intro - introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands available for the DS90 Computer in the D-NIX 5.3 Extension Package. Commands in the D-NIX 5.3 Main Package are described in the *D-NIX 5.3 Reference Manual*.

Manual Page Command Syntax

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

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

where:

[ ]	Surround an *option* or *cmdarg* that is not required.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a " - ".) *noargletter* ... or, *argletter optarg* [,...]
noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one " - " (Rule 5, below).
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or separated by white space and quoted (Rule 8, below).
cmdarg	Path name (or other command argument) *not* beginning with " - ", or " - " by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. *getopts* (1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.
2. Command names must include only lower-case letters and digits.
3. Option names (*option* above) must be one character long.

4. All options must be preceded by " - ".
5. Options with no arguments may be grouped after a single " - ".
6. The first option-argument (*optarg* above) following an option must be preceded by white space.
7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., `-o xxx,z,yy` or `-o "xxx z yy"`).
9. All options must precede operands (*cmdarg* above) on the command line.
10. " -- " may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. " - " preceded and followed by white space should only be used to mean standard input.

SEE ALSO

getopts(1).

exit(2), wait(2), getopt(3C) in the *Programmer's Reference Manual.*

DIAGNOSTICS

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

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

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

DESCRIPTION

acctcom reads file, the standard input, or **/usr/adm/pacct**, in the form described by *acct* (4) 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), STAT (the system exit status), HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD, and BLOCKS READ (total blocks read and written).

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

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

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

-a	Show some average statistics about the processes selected. The statistics will be printed after the output records.
-b	Read backwards, showing latest commands first. This option has no effect when the standard input is read.
-f	Print the fork/exec flag and system exit status columns in the output.
-h	Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as: (total CPU time)/(elapsed time).
-i	Print columns containing the I/O counts in the output.
-k	Instead of memory size, show total kcore-minutes.
-m	Show mean core size (the default).
-r	Show CPU factor (user time/(system-time + user-time).
-t	Show separate system and user CPU times.
-v	Exclude column headings from the output.
-l ***line***	Show only processes belonging to terminal **/dev/line.**

-u *user*	Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a # which designates only those processes executed with super-user privileges, or ? which designates only those processes associated with unknown user IDs.
-g *group*	Show only processes belonging to *group*. The group may be designated by either the group ID or group name.
-s *time*	Select processes existing at or after *time*, given in the format hr[:min[:sec]].
-e *time*	Select processes existing at or before *time*.
-S *time*	Select processes starting at or after *time*.
-E *time*	Select processes ending at or before *time*. Using the same time for both **-S** and **-E** shows the processes that existed at time.
-n *pattern*	Show only commands matching *pattern* that may be a regular expression as in *ed* (1) except that + means one or more occurrences.
-q	Do not print any output records, just print the average statistics as with the **-a** option.
-o *ofile*	Copy selected process records in the input data format to *ofile*; supress standard output printing.
-H *factor*	Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **-h** above.
-O *sec*	Show only processes with CPU system time exceeding *sec* seconds.
-C *sec*	Show only processes with total CPU time, system plus user, exceeding *sec* seconds.
-I *chars*	Show only processes transferring more characters than the cut-off number given by *chars*.

FILES

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

SEE ALSO

ps(1), su(1) in the *D-NIX 5.3 Reference Manual.*

acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual.*

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M) in the *Administrator's Reference Manual.*

BUGS

acctcom only reports on processes that have terminated; use *ps* (1) for active processes. If time exceeds the present time, then time is interpreted as occurring on the previous day.

NAME

at, batch - execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]
at -r *job* ...
at -l [*job* ...]
batch

DESCRIPTION

at and *batch* read commands from standard input to be executed at a later time. *at* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. *at* may be used with the following options:

-r	Removes jobs previously scheduled with *at*.
-l	Reports all jobs scheduled for the invoking user.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

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

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

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

The optional *increment* is simply a number suffixed by one of the following: **minutes, hours, days, weeks, months,** or **years.** (The singular form is also accepted.)

Thus legitimate commands include:

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

at and *batch* write the job number and schedule time to standard error.

batch submits a batch job. It is almost equivalent to "at now", but not quite. For one, it goes into a different queue. For another, "at now" will respond with the error message **too late**.

at **-r** removes jobs previously scheduled by *at* or *batch*. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at* **-l**. You can only remove your own jobs unless you are the super-user.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *sh (1)* provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
sort filename >outfile
<control-D>
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
sort filename 2>&1 >outfile | mail loginid
!
```

To have a job reschedule itself, invoke *at* from within the shell procedure, by including code similar to the following within the shell file:

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

FILES

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

SEE ALSO

kill(1), mail(1), nice(1), ps(1), sh(1), sort(1), cron(1M) in the *D-NIX 5.3 Reference Manual.*

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [**-Fc**] [prog] [parameters] [files]

DESCRIPTION

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

Parameters, in the form x=... y=... etc., may be passed to *awk*.

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

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted **$1, $2,** ...; **$0** refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

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

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

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

The *print* statement prints its arguments on the standard output (or on a file if **> *expr*** is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format [see *printf*(3S) in the *Programmer's Reference Manual*].

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

Patterns are arbitrary Boolean combinations (**!**, **| |**, **&&**, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see *grep* (1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

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

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

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

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

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

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

Print file, filling in page numbers starting at 5:

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

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

SEE ALSO

grep(1), sed(1) in the *D-NIX 5.3 Reference Manual.*

lex(1), printf(3S) in the *Programmer's Reference Manual.*

BUGS

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

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

AWK(1)

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

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

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The *bc* (1) utility is actually a preprocessor for *dc* (1), which it invokes automatically unless the **-c** option is present. In this case the *dc* input is sent to the standard output instead. The options are as follows:

-c	Compile only. The output is send to the standard output.
-l	Argument stands for the name of an arbitrary precision math library.

The syntax for *bc* programs is as follows; `L` means letter a-z, `E` means expression, `S` means statement.

Comments

are enclosed in /* and */.

Names

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

Other operands

arbitrarily long numbers with optional sign and decimal point.
`( E )`
`sqrt ( E )`
`length ( E )` number of significant decimal digits
`scale ( E )` number of digits right of decimal point
`L ( E , ... , E )`

Operators

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

Statements

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

break
quit

Function definitions

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

Functions in -l math library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc* (1). Assignments to *ibase* or *obase* set the input and output number radix respectively.

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

EXAMPLE

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

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

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

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

FILES

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

SEE ALSO

dc(1).

BUGS

The *bc* command does not yet recognize the logical operators, && and | |.

For statement must have all three expressions (E's).

Quit is interpreted when read, not when executed.

BC(1)

NAME

bdiff - big diff

SYNOPSIS

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

DESCRIPTION

bdiff is used in a manner analogous to *diff* (1) to find which lines in two files must be changed to bring the files into agreement. Its purpose is to allow processing of files which are too large for *diff.*

The parameters to *bdiff* are:

file1 (file2)	The name of a file to be used. If *file1 (file2)* is -, the standard input is read.
n	The number of line segments. The value of *n* is 3500 by default. If the optional third argument is given and it is numeric, it is used as the value for *n.* This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail.
-s	Specifies that no diagnostics are to be printed by *bdiff* (silent option). Note, however, that this does not suppress possible diagnostic messages from *diff* (1), which *bdiff* calls.

bdiff ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. If both optional arguments are specified, they must appear in the order indicated above.

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

FILES

/tmp/bd?????

SEE ALSO

diff(1), help(1).

DIAGNOSTICS

Use *help* (1) for explanations.

BDIFF(1)

NAME

bfs - big file scanner

SYNOPSIS

bfs [-] name

DESCRIPTION

The *bfs* command is (almost) like *ed* (1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). *bfs* is usually more efficient than *ed* (1) for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit* (1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional - suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed, as in *ed* (1). Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* (1) are supported. In addition, regular expressions may be surrounded with two symbols besides / and **?** : **>** indicates downward search without wrap-around, and **<** indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

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

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

xn List the marks currently in use (marks are set by the **k** command).

xo [*file*] Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666 (readable and writable by everyone), unless your *umask* setting (see *umask*(1)) dictates otherwise. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

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

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

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

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

2. The second address is less than the first.

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

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

```
xb/^/ label
```

is an unconditional jump.

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

xt ***number*** Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

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

The variable name is the specified *digit* following the **xv**. 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 reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6:**

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

will all print the first 100 lines.

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

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

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

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

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

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

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

```
xv7\!date
```

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

xbz *label*
xbn *label*

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

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

xc [*switch*]

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

SEE ALSO

csplit(1), ed(1), umask(1).

DIAGNOSTICS

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

BFS(1)

NAME

cal - print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and the United States.

EXAMPLES

An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type:

```
cal 9 1752
```

BUGS

The year is always considered to start in January even though this is historically naive.

Beware that "cal 83" refers to the early Christian era, not the 20th century.

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in his or her login directory and sends them any positive results by *mail* (1). Normally this is done daily by facilities in the UNIX operating system.

FILES

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

SEE ALSO

mail(1).

BUGS

Your calendar must be public information for you to get reminder service.

calendar's extended idea of "tomorrow" does not account for holidays.

CALENDAR(1)

NAME

col - filter reverse line-feeds

SYNOPSIS

col [-b] [-f] [-x] [-p]

DESCRIPTION

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

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

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

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

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

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

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

SEE ALSO

nroff(1), tbl(1) in the *DOCUMENTER's WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*

NOTES

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

BUGS

Cannot back up more than 128 lines.

Allows at most 800 characters, including backspaces, on a line.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comm - select or reject lines common to two sorted files

SYNOPSIS

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

DESCRIPTION

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

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm -12** prints only the lines common to the two files; **comm -23** prints only lines in the first file but not in the second; **comm -123** prints nothing.

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1).

COMPRESS(1M)

NAME

compress - compress spell history files

SYNOPSIS

/usr/lib/spell/compress

DESCRIPTION

The command */usr/lib/spell/compress* will strip the **/usr/lib/spell/spellhist** file by removing log information and duplicate occurrences of words from the file.

COMPRESS(1M)

NAME

crypt - encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

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

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

Files encrypted by *crypt* are compatible with those treated by the editors *ed* (1), *edit* (1), *ex* (1), and *vi* (1) in encryption mode.

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

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

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

If the key is an argument to the *crypt* command, it is potentially visible to users executing *ps* (1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), edit(1), ex(1), makekey(1), ps(1), stty(1), vi(1).

WARNING

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

BUGS

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

NAME

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

SYNOPSIS

csh [-cefinstvVxX] [arg ...]

DESCRIPTION

csh is a first implementation of a command language interpreter incorporating a history mechanism (see History Substitutions), job control facilities (see Jobs), interactive file name and user name completion (see File Name Completion), and a C-like syntax. So as to be able to use its job control facilities, users of *csh* must (and automatically) use the new tty driver fully described in *tty* (4). This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See *stty* (1) for details on setting options in the new tty driver.

Note: This implementation does not include the job control facilities.

An instance of *csh* begins by executing commands from the file **.cshrc** in the home directory of the invoker. If this is a login shell then it also executes commands from the file **.login** there. It is typical for users on crt's to put the command **'stty crt'** in their .login file, and to also invoke *tset* (1) there.

In the normal case, the shell will then begin reading commands from the terminal, prompting with % . Processing of arguments and the use of the shell to process files containing command scripts will be described later.

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

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

Lexical structure

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

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

When the shell's input is not a terminal, the character # introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by \ and in quotations using `, ', and ".

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by | characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ;, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an **&**.

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

Jobs

The shell associates a job with each pipeline. It keeps a table of current jobs, printed by the jobs command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been **Stopped**, and print another prompt. You can then manipulate the state of this job, putting it in the background with the bg command, or run some other commands and then eventually bring the job back into the foreground with the foreground command fg. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is another special key ^Y which does not generate a STOP signal until a program attempts to *read* (2) it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command **'stty tostop'**. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character % introduces a job name. If you wish to refer to job number 1, you can name it as **%1**. Just naming a job brings it to the foreground; thus **%1** is a synonym for **fg %1**, bringing job 1 back into the foreground. Similarly saying **%1 &** resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous, thus **%ex** would normally restart a suspended *ex* (1) job, if there were only one suspended job whose name began with the string **ex**. It is also possible to say

say **%?string** which specifies a job whose text contains string, if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a -. The abbreviation **%+** refers to the current job and **%-** refers to the previous job. For close analogy with the syntax of the history mechanism (described below), **%%** is also a synonym for the current job.

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable notify, the shell will notify you immediately of changes of status in background jobs. There is also a shell command notify which marks a single process so that its status changes will be immediately reported. By default notify marks the current process; simply say **notify** after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that **You have stopped jobs.** You may use the jobs command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.

File Name Completion

When the file name completion feature is enabled by setting the shell variable filec (see set), *csh* will interactively complete file names and user names from unique prefixes, when they are input from the terminal followed by the escape character (the escape key, or control-[). For example, if the current directory looks like

```
DSC.OLD     bin       cmd       lib       xmpl.c
DSC.NEW     chaosnet  cmtest    mail      xmpl.o
bench       class     dev       mbox      xmpl.out
```

and the input is

```
% vi ch<escape>
```

csh will complete the prefix **'ch'** to the only matching file name **'chaosnet'**, changing the input line to

```
% vi chaosnet
```

However, given

```
% vi D<escape>
```

csh will only expand the input to

```
% vi DSC.
```

and will sound the terminal bell to indicate that the expansion is incomplete, since there are two file names matching the prefix **'D'**.

If a partial file name is followed by the end-of-file character (usually control-D), then, instead of completing the name, *csh* will list all file names matching the prefix. For example, the input

```
% vi D<control-D>
```

causes all files beginning with 'D' to be listed:

```
DSC.NEW DSC.OLD
```

while the input line remains unchanged.

The same system of escape and end-of-file can also be used to expand partial user names, if the word to be completed (or listed) begins with the character ~. For example, typing

```
cd ~ro<escape>
```

may produce the expansion

```
cd ~root
```

The use of the terminal bell to signal errors or multiple matches can be inhibited by setting the variable nobeep.

Normally, all files in the particular directory are candidates for name completion. Files with certain suffixes can be excluded from consideration by setting the variable fignore to the list of suffixes to be ignored. Thus, if fignore is set by the command

```
% set fignore = (.o .out)
```

then typing

```
% vi x<escape>
```

would result in the completion to

```
% vi xmpl.c
```

ignoring the files "xmpl.o" and "xmpl.out". However, if the only completion possible requires not ignoring these suffixes, then they are not ignored. In addition, fignore does not affect the listing of file names by control-D. All files are listed regardless of their suffixes.

Substitutions

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

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character ! and may begin anywhere in the input stream (with the proviso that they do not nest.) This ! may be preceded by an \ to prevent its special meaning; for convenience, a ! is passed unchanged when it is followed by a blank, tab, newline, = or (. (History substitutions also occur when an input line begins with ^. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences

of words from these saved commands into the input stream. The size of which is controlled by the history variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

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

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

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

With the current event 13 we can refer to previous events by event number **!11**, relatively as in **!-2** (referring to the same event), by a prefix of a command word as in **!d** for event 12 or **!wri** for event 9, or by a string contained in a word in the command as in **!?mic?** also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case !! refers to the previous command; thus !! alone is essentially a redo.

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

0	first (command) word
n	n'th argument
^	first argument, i.e. **1**
$	last argument
%	word matched by (immediately preceding) ?s? search
x-y	range of words
-y	abbreviates **0-y**
*	abbreviates **^-$**, or nothing if only 1 word in event
x*	abbreviates **x-$**
x-	like **x*** but omitting word **$**

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

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing **.xxx** component, leaving the root name.
e	Remove all but the extension **.xxx** part.
s/l/r/	Substitute l for r

t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. **g&**.
p	Print the new command line but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a **g** the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

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

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

A special abbreviation of a history reference occurs when the first nonblank character of an input line is a ^. This is equivalent to **!:s^** providing a convenient shorthand for substitutions on the text of the previous line. Thus **^lb^lib** fixes the spelling of **lib** in the previous command. Finally, a history substitution may be surrounded with { and } if necessary to insulate it from the characters which follow. Thus, after **ls -ld ~paul** we might do **!{l}a** to do **ls -ld ~paula**, while **!la** would look for a command starting **la**.

Quotations with ' and "

The quotation of strings by ' and " can be used to prevent all or some of the remaining substitutions. Strings enclosed in ' are prevented any further interpretation. Strings enclosed in " may be expanded as described below.

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

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the alias and unalias commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history

mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

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

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

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

Variable substitution

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

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

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

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

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

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

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

$name
${name}

Are replaced by the words of the value of variable name, each separated by a blank. Braces insulate name from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter. If name is not a shell variable, but is set in the environment, then that value is returned (but : modif iers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}

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

$#name
${#name}

Gives the number of words in the variable. This is useful for later use in a **[selector]**.

$0

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

$number
${number}

Equivalent to **$argv[number]**.

$*

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

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

The following substitutions may not be modified with : modifiers.

$?name
${?name}

Substitutes the string **1** if name is set, **0** if it is not.

$?0

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

$$

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

$<

Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

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

Command substitution

Command substitution is indicated by a command enclosed in ‘. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within "s, only newlines force new words; blanks and tabs are preserved.

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

Filename substitution

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

In matching filenames, the character . at the beginning of a filename or immediately following a /, as well as the character / must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence [...]

matches any one of the characters enclosed. Within [...], a pair of characters separated by - matches any character lexically between the two.

The character ~ at the beginning of a filename is used to refer to home directories. Standing alone, i.e. ~ it expands to the invokers home directory as reflected in the value of the variable home. When followed by a name consisting of letters, digits and - characters the shell searches for a user with that name and substitutes their home directory; thus **~ken** might expand to **/usr/ken** and **~ken/chmach** to **/usr/ken/chmach**. If the character ~ is followed by a character other than a letter or / or appears not at the beginning of a word, it is left undisturbed.

The metanotation **a{b,c,d}e** is a shorthand for **abe ace ade**. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus **~source/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/ls.c** whether or not these files exist without any chance of error if the home directory for **source** is **/usr/source**. Similarly **../{memo,*box}** might expand to **../memo ../box ../mbox**. (Note that **memo** was not sorted with the results of matching ***box**.) As a special case {, } and {} are passed undisturbed.

Input/output

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

< name

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

<< word

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

> name
>! name
>& name
>&! name

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

If the variable noclobber is set, then the file must not exist or be a character special file (e.g. a termi nal or **/dev/null**) or an error results.

This helps prevent accidental destruction of files. In this case the ! forms can be used and suppress this check.

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

>> name
>>& name
>>! name
>>&! name

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

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is not modified to be the empty file **/dev/null**; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see Jobs above).

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

Expressions

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

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

Here the precedence increases to the right, == != =~ and !~, <= >= < and >, << and >>, + and -, * / and % being, in groups, at the same level. The == != =~ and !~ operators compare their arguments as strings; all others operate on numbers. The operators =~ and !~ are like != and == except that the right hand side is a pattern (containing, e.g. *s, ?s and instances of [...]) against which the left hand operand is matched. This reduces the need for use of the switch statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with **0** are considered octal numbers. Null or missing arguments are considered **0**. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser (& | < > ( )) they should be surrounded by spaces.

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

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

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. **0**. Command executions succeed, returning true, i.e. **1**, if the command exits with status 0, otherwise they fail, returning false, i.e. **0**. If more detailed status information is required then the command should be executed outside of an expression and the variable status examined.

Control flow

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

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

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

Builtin commands

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

alias
alias name
alias name wordlist

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

alloc

Shows the amount of dynamic memory acquired, broken down into used and free memory. With an argument shows the number of free and used blocks in each size category. The categories start at size 8 and double at each step. This command's output may vary across system types, since systems other than the VAX may use a different memory allocator.

bg
bg %job ...

Puts the current or specified jobs into the background, continuing them if they were stopped.

break

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

breaksw

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

case label:

A label in a switch statement as discussed below.

cd
cd name
chdir
chdir name

Change the shell's working directory to directory name. If no argument is given then change to the home directory of the user. If name is not found as a subdirectory of the current directory (and does not begin with /, ./ or ../), then each component of the variable cdpath is checked to see if it has a subdirectory name. Finally, if all else fails but name is a shell variable whose value begins with /, then this is tried to see if it is a directory.

continue

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

default:

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

dirs

Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist
echo -n wordlist

The specified words are written to the shells standard output, separated by spaces, and terminated with a new line unless the -n option is specified.

else
end
endif
endsw

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

eval arg ...

(As in *sh* (1).) The arguments are read as input to the shell and the resulting command(s) executed in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset* (1) for an example of using eval.

exec command

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

exit
exit(expr)

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

fg
fg %job ...

Brings the current or specified jobs into the foreground, continuing them if they were stopped.

foreach name (wordlist)
...
end

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

The builtin command continue may be used to continue the loop prematurely and the builtin command break to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with ? before any statements in the loop are exe cuted. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist

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

goto word

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

hashstat

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding exec's). An exec is attempted for each component of the path where the hash function indicates a possible hit, and in each component which does not begin with a /.

history
history n
history -r n
history -h n

Displays the history event list; if n is given only the n most recent events are printed. The **-r** option reverses the order of printout to be most recent first rather than oldest first. The **-h** option causes the history list to be printed without leading numbers. This is used to produce files suitable for sourceing using the **-h** option to source.

if (expr) command

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

if (expr) then
...
else if (expr2) then
...
else
...
endif

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

jobs
jobs -l

Lists the active jobs; given the **-l** options lists process id's in addition to the normal information.

kill %job
kill -sig %job ...
kill pid
kill -sig pid ...
kill -l

Sends either the **TERM** (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in **/usr/include/signal.h**, stripped of the prefix **SIG**). The signal names are listed by **kill -l**. There is no default, saying just **kill** does not send a signal to the current job. If the signal being sent is **TERM** (terminate) or **HUP** (hangup), then the job or process will be sent a **CONT** (continue) signal as well.

limit
limit resource
limit resource maximum-use
limit -h
limit -h resource
limit -h resource maximum-use

Limits the consumption by the current process and each process it creates to not individually exceed maximum use on the specified resource. If no maximum-use resource is given, then all limitations are given. If the **-h** flag is given, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. Only the super-user may raise the hard limits, but a user may lower or raise the current limits within the legal range.

Resources controllable currently include cputime (the maximum number of cpu-seconds to be used by each pro cess), filesize (the largest single file which can be created), datasize (the maximum growth of the data+stack region via *sbrk* (2) beyond the end of the program text), stacksize (the maximum size of the automatically-extended stack region), and coredumpsize (the size of the largest core dump that will be created).

The maximum-use may be given as a (floating point or integer) number followed by a scale factor. For all limits other than cputime the default scale is **k** or **kilobytes** (1024 bytes); a scale factor of **m** or **megabytes** may also be used. For cputime the default scaling is **seconds**, while **m** for minutes or **h** for hours, or a time of the form **mm:ss** giving minutes and seconds may be used.

For both resource names and scale factors, unambiguous prefixes of the names suffice.

login

Terminate a login shell, replacing it with an instance of */bin/login*. This is one way to log off, included for compatibility with *sh* (1).

logout

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

nice
nice +number
nice command
nice +number command

The first form sets the scheduling priority for this shell to 4. The second form sets the priority to the given number. The final two forms run command at priority 4 and number respectively. The greater the number, the less cpu the process will get. The superuser may specify negative priority by using **nice -number ....** Command is always executed in a subshell, and the restrictions placed on commands in sim ple if statements apply.

nohup
nohup command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with **&** are effectively nohup'ed.

notify
notify %job ...

Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable notify is set.

onintr
onintr -
onintr label

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

In any case, if the shell is running detached and interrupts are being ignored, all forms of onintr have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd
popd +n

Pops the directory stack, returning to the new top directory. With an argument **+n** discards the nth entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd
pushd name
pushd +n

With no arguments, pushd exchanges the top two elements of the directory stack. Given a name argument, pushd changes to the new directory (ala *cd*) and pushes the old current working directory (as in *csw*)

onto the directory stack. With a numeric argument, rotates the nth argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash

Causes the internal hash table of the contents of the directories in the path variable to be recomputed. This is needed if new commands are added to directories in the path while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified command which is subject to the same restrictions as the command in the one line if statement above, is executed count times. I/O redirections occur exactly once, even if count is 0.

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

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

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

setenv
setenv name value
setenv name

The first form lists all current environment variables. The last form sets the value of environment variable name to be value, a single string. The second form sets name to an empty string. The most commonly used environment variable **USER**, **TERM**, and **PATH** are automatically imported to and exported from the *csh* variables user, term, and path; there is no need to use setenv for these.

shift
shift variable

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

source name
source -h name

The shell reads commands from name. Source commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a source at any level terminates all nested source commands. Normally input during source commands is not placed on the history list; the -h option causes the commands to be placed in the history list without being executed.

stop
stop %job ...

Stops the current or specified job which is executing in the background.

suspend

Causes the shell to stop in its tracks, much as if it had been sent a stop signal with ^Z. This is most often used to stop shells started by *su* (1).

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

Each case label is successively matched, against the specified string which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a **default** label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command breaksw causes execution to continue after the endsw. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the endsw.

time
time command

With no argument, a summary of time used by this she the specified simple command is timed and a time summary as described under the time variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern

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

unhash

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

unlimit
unlimit resource
unlimit -h
unlimit -h resource

Removes the limitation on resource. If no resource is specified, then all resource limitations are removed. If -h is given, the corresponding hard limits are removed. Only the super-user may do this.

unset pattern

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

unsetenv pattern

Removes all variables whose name match the specified pattern from the environment. See also the setenv command above and *printenv* (1).

wait

All background jobs are waited for. It the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end

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

%job

Brings the specified job into the foreground.

%job &

Continues the specified job in the background.

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

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

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

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

Pre-defined and environment variables

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

This shell copies the environment variable **USER** into the variable user, **TERM** into term, and **HOME** into home, and copies these back into the environment whenever the normal shellvariables are reset. The environment variable **PATH** is likewise handled; it is not necessary to worry about its setting other than in the file .cshrc as inferior csh processes will import the definition of path from the environment, and re-export it if you then change it.

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. **$1** is replaced by **$argv[1]**, etc.

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

cwd The full pathname of the current directory.

echo Set when the **-x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selec tively.

filec Enable file name completion.

histchars Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character !. The second character

of its value replaces the character ^ in quick substitutions.

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

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

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.

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

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

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

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

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

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

notify If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a prompt.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no path variable then only full path names will execute. The usual search path is ., **/bin** and **/usr/bin**, but this may vary from system to system. For the super-user the

default search path is **/etc**, **/bin** and **/usr/bin**. A shell which is given neither the **-c** nor the **-t** option will normally hash the contents of the directories in the path variable after reading **.cshrc**, and each time the path variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the rehash or the commands may not be found.

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

savehist is given a numeric value to control the number of entries of the history list that are saved in **~/.history** when the user logs out. Any command which has been referenced in this many events will be saved. During start up the shell sources **~/.history** into the history list enabling history to be saved across logins. Too large values of savehist will slow down the shell during start up.

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

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status **1**, all other builtin commands set status **0**.

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

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

Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via *execve* (2). Each word in the variable path names a directory from which the shell will attempt to execute the command. If it is given neither a **-c** nor a **-t** option, the shell will hash the names in these directories into an internal table so that it will only try an exec in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via unhash), or if the shell was given a **-c** or **-t** argument, and

in any case for each directory component of path which does not begin with a /, the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus **(cd ; pwd) ; pwd** prints the home directory; leaving you where you were (printing this after the home directory), while **cd ; pwd** leaves you in the home directory. Parenthesized commands are most often used to prevent chdir from affecting the current shell.

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

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

Argument list processing

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

-b This flag forces a **'break'** from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.

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

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

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

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

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

-s Command input is taken from the standard input.

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

-v	Causes the verbose variable to be set, with the effect that command input is echoed after history substitution.
-x	Causes the echo variable to be set, so that commands are echoed immediately before execution.
-V	Causes the verbose variable to be set even before **.cshrc** is executed.
-X	Is to **-x** as **-V** is to **-v**.

After processing of flag arguments, if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by **$0**. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a **standard** shell if the first character of a script is not a #, i.e. if the script does not start with a comment. Remaining arguments initialize the variable argv.

Signal handling

The shell normally ignores quit signals. Jobs running detached (either by & or the bg or %... & commands) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by onintr. Login shells catch the terminate signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file **.logout**.

AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp of I.I.A.S.A, Laxenburg, Austria, with different syntax than that used now. File name completion code written by Ken Greer, HP Labs.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after **.cshrc** at login.
~/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a #.
/tmp/sh*	Temporary file for **<<**.
/etc/passwd	Source of home directories for **~name**.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of alias substitutions on a single line to 20.

This implementation does not include the job control facilities.

SEE ALSO

sh(1), access(2), execve(2), fork(2), killpg(2), pipe(2), sigvec(2), umask(2), setrlimit(2), wait(2), tty(4), a.out(5), environ(7).

BUGS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (i.e. wrong) as the job may have changed directories internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the form **a ; b ; c** are also not handled gracefully when stopping is attempted. If you suspend **b**, the shell will then immediately execute **c**. This is especially noticeable if this expansion results from an alias. It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. **(a ; b ; c)**.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by **?**, are not placed in the history list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with **|**, and to be used with **&** and **;** metasyntax.

It should be possible to use the **:** modifiers on the output of command substitutions. All and more than one **:** modifier should be allowed on **$** substitutions. The way the filec facility is implemented is ugly and expensive.

NAME

csplit - context split

SYNOPSIS

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

DESCRIPTION

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

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

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

.
.
.

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

If the *file* argument is a - then standard input is used.

The options to *csplit* are:

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

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

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

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

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

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

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

{num} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows

lnno,the file will be split every *lnno* lines (*num* times) from that point.

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

EXAMPLES

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

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

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

Note that this example overwrites the original file.

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

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

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

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

SEE ALSO

ed(1), sh(1).

regexp(5) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Self-explanatory except for:

```
arg - out of range
```

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

NAME

ctags - create a tags file

SYNOPSIS

ctags [**-BFatuwvx**] [**-f** *tagsfile*] name ...

DESCRIPTION

ctags makes a tags file for *ex* (1) from the specified C, Pascal, Fortran, YACC, lex, and lisp sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these objects definitions.

If the **-x** flag is given, *ctags* produces a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

If the **-v** flag is given, an index of the form expected by *vgrind* (1) is produced on the standard output. This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through **sort -f.** Sample use:

```
ctags -v files | sort -f  index
vgrind -x index
```

Normally *ctags* places the tag descriptions in a file called *tags*; this may be overridden with the **-f** option.

Files whose names end in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Files whose names end in **.y** are assumed to be YACC source files. Files whose names end in **.l** are assumed to be either lisp files if their first non-blank character is ';', '(', or '[', or lex files otherwise. Other files are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:

-F	use forward searching patterns (/.../) (default).
-B	use backward searching patterns (?...?).
-a	append to tags file.
-t	create tags for typedefs.
-w	suppressing warning diagnostics.
-u	causing the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is imple-

mented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

FILES

tags	output tags file

SEE ALSO

ex(1), vi(1)

AUTHOR

Ken Arnold; FORTRAN added by Jim Kleckner; Bill Joy added Pascal and **-x**, replacing *cxref;* C typedefs added by Ed Pelegri-Llopart.

BUGS

Recognition of **functions, subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

Does not know about #ifdefs.

Should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **-tx** shows only the last line of typedefs.

NAME

dc - desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See *bc* (1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *bc* also provides reasonable control structures for programs.) The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number	The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.
+ - / * % ^	The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.
s*x*	The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.
l*x*	The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.
d	The top value on the stack is duplicated.
p	The top value on the stack is printed. The top value remains unchanged.
P	Interprets the top of the stack as an ASCII string, removes it, and prints it.
f	All values on the stack are printed.
q	Exits the program. If executing a string, the recursion level is popped by two.
Q	Exits the program. The top value on the stack is popped and the string execution level is popped by that value.
x	Treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X	Replaces the number on the top of the stack with its scale factor.
[...]	Puts the bracketed ASCII string onto the top of the stack.
< x > x = x	The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.
v	Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.
!	Interprets the rest of the line as a UNIX system command.
c	All values on the stack are popped.
i	The top value on the stack is popped and used as the number radix for further input. I Pushes the input base on the top of the stack.
o	The top value on the stack is popped and used as the number radix for further output.
O	Pushes the output base on the top of the stack.
k	The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
z	The stack level is pushed onto the stack.
Z	Replaces the number on the top of the stack with its length.
?	A line of input is taken from the input source (usually the terminal) and executed.
; :	are used by *bc* (1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

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

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented

where *x* is an octal number.

stack empty

for not enough elements on the stack to do what was asked.

Out of space

when the free list is exhausted (too many digits).

Out of headers

for too many numbers being kept around.

Out of pushdown

for too many items on the stack.

Nesting Depth

for too many levels of nested execution.

DC(1)

NAME

deroff - remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [-mx] [-w] [files]

DESCRIPTION

deroff reads each of the *files* in sequence and removes all *troff* (1) requests, macro calls, backslash constructs, *eqn* (1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl* (1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **-m** option may be followed by an **m**, **s**, or **l**. The **-mm** option causes the macros to be interpreted so that only running text is output (i.e., no text from macro lines.) The **-ml** option forces the **-mm** option and also causes deletion of lists associated with the **mm** macros.

If the **-w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (**&**), and apostrophes (**'**); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1) in the *DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*

BUGS

deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The **-ml** option does not handle nested lists correctly.

DEROFF(1)

NAME

diff - differential file comparator

SYNOPSIS

diff [**-efbh**] file1 file2

DESCRIPTION

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

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

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

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

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

The **-e** option produces a script of *a*, *c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **-f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

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

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

Option **-h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **-e** and **-f** are unavailable with **-h**.

FILES

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

SEE ALSO

bdiff(1), cmp(1), comm(1), ed(1).

DIAGNOSTICS

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

BUGS

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

WARNINGS

Missing newline at end of file X

indicates that the last line of file **X** did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

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

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

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

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

Under the **-e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged **====** and **====3**. Option **-x** (**-3**) produces a script to incorporate only changes flagged **====** (**====3**). The following command will apply the resulting script to *file1*.

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

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

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

Files longer than 64K bytes will not work.

DIFF3(1)

NAME

diffmk - mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff* or *troff* (1). *name1* and *name2* are the old and new versions of the file. *diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file macs contains:

```
.pl 1      .ll 77      .nf      .eo      .nc
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shellprocedure).

SEE ALSO

diff(1), nroff(1), troff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

DIFFMK(1)

NAME

dircmp - directory comparison

SYNOPSIS

dircmp [**-d**] [**-s**] [**-w***n*] dir1 dir2

DESCRIPTION

dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

-d	Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff* (1).
-s	Suppress messages about identical files.
-w*n*	Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1).

NAME

edit - text editor (variant of ex for casual users)

SYNOPSIS

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

DESCRIPTION

edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor.

-r Recover file after an editor or system crash.

-x Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see *crypt* (1)). Also, see the WARNING section at the end of this manual page.

The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

To edit the contents of an existing file you begin with the command "edit name" to the shell. *edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete**, the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say **s**/*old*/*new*/ where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say [Modified] if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No write since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a .). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., **3,5change**. You can print lines this way too. Thus **1,23p** prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command **z.**. The current line will then be the last line printed; you can get back to the line where you were before the **z.** command by saying ". The **z** command can also be given other following characters **z-** prints a screen of text (or 24 lines) ending where you are; **z+** prints the next screenful. If you want less than a screenful of lines, type in **z.12** to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command **delete 5**.

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this

is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form **/text/** to search forward for *text* or **?text?** to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form **/^text/** which searches for *text* at the beginning of a line. Similarly **/text$/** searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name .; this is most useful in a range of lines as in **.,$print** which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name **$**. Thus the command **$ delete** or **$d** deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line **$-5** is the fifth before the last, and **.+20** is 20 lines after the present.

You can find out which line you are at by doing **.=**. This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say **10,20delete a** which deletes these lines from the file and places them in a buffer named *a*. *edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing **put a** to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines, following it with the name of the other file you wish to edit, i.e., **edit chapter2**. By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say **10,20move $** for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ed(1), ex(1), vi(1).

EDIT(1)

NAME

egrep - search a file for a pattern using full regular expressions

SYNOPSIS

egrep [options] full regular expression [file ...]

DESCRIPTION

egrep (*expression grep*) searches files for a pattern of characters and prints all lines that contain that pattern. *egrep* uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

egrep accepts full regular expressions as in *ed* (1), except for \(and \),with the addition of:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses () for grouping.

Be careful using the characters **$**, *, [, ^, |, (,), and \ in *full regular expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes '...'.

The order of precedence of operators is [], then *?+, then concatenation, then | and new-line.

If no files are specified, *egrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
-c	Print only a count of the lines that contain the pattern.
-i	Ignore upper/lower case distinction during comparisons.
-l	Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
-n	Precede each line by its line number in the file (first line is 1).
-v	Print all lines except those that contain the pattern.

-e *special_expression*

Search for a *special expression* (*full regular expression* that begins with a -).

-f *file* Take the list of *full regular expressions* from *file.*

SEE ALSO

ed(1), fgrep(1), grep(1), sed(1), sh(1).

DIAGNOSTICS

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

BUGS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h.**

NAME

eqn, neqn, checkeq - typeset mathematics

SYNOPSIS

eqn [**-d***xy*] [**-p***n*] [**-s***n*] [**-f***n*] [file] ...
checkeq [file] ...

DESCRIPTION

eqn is a *troff* (1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, *neqn* on terminals. Usage is almost always

```
eqn file ... | troff
neqn file ... | nroff
```

If no files are specified, these programs read from the standard input. A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a line beginning with .EN. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **-d***xy* or (more commonly) with 'delim *xy* ' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Fractions are made with **over**.

sqrt makes square roots.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile, lpile, cpile,** and **rpile**. There can be an arbitrary number of elements in a **pile**. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot, dotdot, hat, tilde, bar, vec, dyad,** and **under.**

Sizes and font can be changed with **size** *n* or **size** ±n, **roman, italic, bold,** and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **-s n** and **-f n.**

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **-p n.**

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define** : *define thing % replacement %* defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The *%* may be any character that does not occur in *replacement.*

Keywords like *sum* (∑) *int* (∫) *inf* (∞) and shorthands like >= (≥) -> (→), and != (≠) are recognized. Greek letters are spelled out in the desired case, as in *alpha* or *GAMMA*. Mathematical words like sin, cos, log are made Roman automatically. *troff* (1) four-character escapes like \(bs can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

SEE ALSO

tbl(1), ms(7), eqnchar(7)

B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics - User's Guide*

J. F. Ossanna, *NROFF/TROFF User's Manual*

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in 'bold "12.3"'.

NAME

ex - text editor

SYNOPSIS

ex [-] [**-v**] [**-t** *tag*] [**-r**] [**-R**] [**-x**] [*+command*] *name* ...

DESCRIPTION

ex is the root of a family of editors: *ex* and *vi*. *ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi* (1), which is a command which focuses on the display editing portion of *ex*.

For ed Users

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base (see *terminfo* (4)) and the type of the terminal you are using from the variable **TERM** in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi* (1).

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

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

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

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the **^D** key as a backtab and space and tab forward to align new code easily.

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

INVOCATION OPTIONS

The following invocation options are interpreted by *ex :*

-	Suppress all interactive-user feedback. This is useful in processing editor scripts.
-v	Invokes *vi*
t *tag*	Edit the file containing the *tag* and position the editor at its definition.
r *file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
-R	*Readonly* mode set, prevents accidentally overwriting the file.
-x	Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see *crypt* (1)). Also, see the WARNING section at the end of this manual page.
+ ***command***	Begin editing by executing the specified editor search or positioning *command*.

The *name* argument indicates files to be edited.

ex States

Command Normal and initial state. Input prompted for by :. Your kill character cancels partial command.

Insert Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert is normally terminated by a line having only . on it, or abnormally with an interrupt.

Visual Entered by **vi**, terminates with **Q** or ^\.

ex command names and abbreviations

abbrev	ab	next	n	unabbrev	una
append	a	number	nu	undo	u
args	ar			unmap	unm
change	c	preserve	pre	version	ve
copy	co	print	p	visual	vi
delete	d	put	pu	write	w
edit	e	quit	q	xit	x
file	f	read	re	yank	ya
global	g	recover	rec	window	z
insert	i	rewind	rew	escape	!
join	j	set	se	lshift	<
list	l	shell	sh	print next	CR
map		source	so	resubst	&
mark	ma	stop	st	rshift	>
move	m	substitute	s	scroll	^D

ex Command Addresses

.n	line *n*	/	next with *pat*
.	current	?	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
-	previous	*'x*	marked with *x*
+n	*n* forward	"	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment var.
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option
set no*x*	disable option
set *x*=	give value *val*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
list		print ^I for tab, $ at end
magic	. [*	special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...

redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in *vi*
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any char in *str*
['str]	... not in *str*
[*x*-*y*]	... between *x* and *y* * any number of preceding

AUTHOR

vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/usr/lib/*/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve/*login*	preservation directory (where *login* is the user's login)

SEE ALSO

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

curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual.*

BUGS

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

undo never clears the buffer modified condition.

The **z** command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '-' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

factor - obtain the prime factors of a number

SYNOPSIS

factor [integer]

DESCRIPTION

When you use *factor* without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. *factor* exits if it encounters a zero or any non-numeric character.

If you invoke *factor* with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. *factor* will take this time when n is prime or the square of a prime.

DIAGNOSTICS

factor prints the error message, "Ouch," for input out of range or for garbage input.

NAME

file - determine file type

SYNOPSIS

file [-c] [-f ffile **] [-m** mfile **]** arg ...

DESCRIPTION

file performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0.

-c The **-c** option causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **-c.**

-f If the **-f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

-m The **-m** option instructs *file* to use an alternate magic file.

file uses the file **/etc/magic** to identify files that have some sort of *magic number,* that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

FILES

/etc/magic

SEE ALSO

filehdr(4) in the *Programmer's Reference Manual.*

FILE(1)

NAME

glossary - definitions of common UNIX system terms and symbols

SYNOPSIS

[**help**] **glossary** [term]

DESCRIPTION

The UNIX system Help Facility command *glossary* provides definitions of common technical terms and symbols.

Without an argument, *glossary* displays a menu screen listing the terms and symbols that are currently included in *glossary*. A user may choose one of the terms or may exit to the shell by typing q (for "quit"). When a term is selected, its definition is retrieved and displayed. By selecting the appropriate menu choice, the list of terms and symbols can be redisplayed.

A term's definition may also be requested directly from shell level (as shown above), causing a definition to be retrieved and the list of terms and symbols not to be displayed. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following is a table which list the symbols and their escape sequence.

SYMBOL	ESCAPE SEQUENCE
""	\"\"
”	\’\’
.	\\.
[]	\\[\\]
“	\‘\‘
#	\#
&	\&
*	*
\	\\\\
|	\\|

From any screen in the Help Facility, a user may execute a command via the shell (*sh* (1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see *profile* (4)): "`export SCROLL ; SCROLL=no' '`. If you later decide that scrolling is desired, **SCROLL** must be set to **yes** .

Information on each of the Help Facility commands (*starter, locate, usage, glossary,* and *help*) is located on their respective manual pages.

SEE ALSO

help(1), helpadm(1M), locate(1), sh(1), starter(1), usage(1).

term(5) in the *Programmer's Reference Manual.*

WARNINGS

If the shell variable **TERM** (see *sh* (1)) is not set in the user's **.profile** file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term* (5).

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 characterr 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

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 kbytes (1024 bytes). Note that this is the one case where "b" does not stand for bytes. Since specifying a hexadecimal offset in blocks would result in an ambiguous trailing 'b', any offset and multiplier may be separated by an asterisk (*).

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

Format Flags

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

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

The character format prints 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 will be broken up. Control characters in the range 0x00 to 0x1f are printed as ^@ to ^_. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special characters (^, ~, \) are preceded by a backslash (\) to escape their special meaning. As special cases, two values are represented numerically as \177 and \377. This flag will override all output format specifiers except addresses.

NAME

help - UNIX system Help Facility

SYNOPSIS

help
[help] starter
[help] usage [-d] [-e] [-o] [command_name **]**
[help] locate [keyword1 **[** keyword2 **] ...]**
[help] glossary [term **]**
help arg ...

DESCRIPTION

The UNIX system Help Facility provides on-line assistance for UNIX system users, whether they desire general information or specific assistance for use of the Source Code Control System (SCCS) commands.

Without arguments, *help* prints a menu of available on-line assistance commands with a short description of their functions. The commands and their descriptions are:

COMMAND	DESCRIPTION
starter	information about the UNIX system for the beginning user
locate	locate UNIX system commands using function-related keywords
usage	UNIX system command usage information
glossary	definitions of UNIX system technical terms

The user may choose one of the above commands by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

With arguments, *help* directly invokes the named on-line assistance command, bypassing the initial *help* menu. The commands *starter, locate, usage,* and *glossary,* optionally preceded by the word *help,* may also be specified at shell level. When executing *glossary* from shell level some of the symbols listed in the glossary must be escaped (preceded by one or more backslashes, " \") to be understood by the Help Facility. For a list of symbols refer and how many backslashes to use for each, refer to the *glossary* (1) manual page.

From any screen in the Help Facility, a user may execute a command via the shell (*sh* (1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable SCROLL must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to

your **.profile** file (see *profile* (4)): ''export SCROLL ; SCROLL=no''. If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter, locate, usage, glossary,* and *help*) is located on their respective manual pages.

The Help Facility can be tailored to a customer's needs by use of the *helpadm* (1M) command.

If the first argument to *help* is different from *starter*, *usage*, *locate*, or *glossary*, *help* assumes information is being requested about the sccs Facility. The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type1	Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., ge3 for message 3 from the *get* command).
type2	Does not contain numerics (as a command, such as *get*).
type3	Is all numeric (e.g., 212).

SEE ALSO

glossary(1), helpadm(1M), locate(1), sh(1), starter(1), usage(1).

admin(1), cdc(1), comb(1), delta(1), get(1), prs(1), rmdel(1), sact(1), sccsdiff(1), unget(1), val(1), vc(1), what(1), profile(4), sccsfile(4), term(5) in the *Programmer's Reference Manual.*

WARNINGS

If the shell variable TERM (see *sh* (1)) is not set in the user's **.profile** file, then TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term* (5).

NAME

helpadm - make changes to the Help Facility database

SYNOPSIS

/etc/helpadm

DESCRIPTION

The UNIX system Help Facility Administration command, *helpadm,* allows UNIX system administrators and command developers to define the content of the Help Facility database for specific commands and to monitor use of the Help Facility. The *helpadm* command can only be executed by login root, login bin, or a login that is a member of group bin.

The *helpadm* command prints a menu of 3 types of Help Facility data which can be modified, and 2 choices relating to monitoring use of the Help Facility. The five choices are:

- modify *startup* data
- add, modify, or delete a *glossary* term
- add, modify, or delete command data (description, options, examples, and keywords)
- prevent monitoring use of the Help Facility (login root and login bin only)
- permit monitoring use of the Help Facility (login root and login bin only)

The user may make one of the above choices by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

If one of the first three choices is chosen, then the user is prompted for additional information; specifically, which *startup* screen, *glossary* term definition, or command description is to be modified. The user may also be prompted for information to identify whether the changes to the database are additions, modifications, or deletions. If the user is modifying existing data or adding new data, then they are prompted to make the appropriate modifications/additions. If the user is deleting a *glossary* term or a command from the database, then they must respond affirmatively to the next query in order for the deletion to be done. In any case, before the user's changes are final, they must respond affirmatively when asked whether they are sure they want their requested database changes to be done.

By default, *helpadm* will put the user into *ed* (1) to make additions/modifications to database information. If the user wishes to be put into a different editor, then they should set the environment variable EDITOR in their environment to the desired editor, and then export EDITOR.

If the user chooses to monitor/prevent monitoring use of the Help Facility, the choice made is acted on with no further interaction by the user.

SEE ALSO

ed(1), glossary(1), help(1), locate(1), starter(1), usage(1).

WARNINGS

When the UNIX system is delivered to a customer, **/etc/profile** exports the environment variable **LOGNAME**. If **/etc/profile** has been changed so that **LOGNAME** is not exported, then the options to monitor/prevent monitoring use of the Help Facility may not work properly.

FILES

HELPLOG	/usr/lib/help/HELPLOG
helpclean	/usr/lib/help/helpclean

NAME

id - print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

id outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1) in the *D-NIX 5.3 Reference Manual.*

getuid(2) in the *Programmer's Reference Manual.*

NAME

join - relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

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

file1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line [see *sort* (1)].

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*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a 1 or a 2 referring to either *file1* or *file2*, respectively. The following options are recognized:

-a*n*	In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
-e*s*	Replace empty output fields by string *s*.
-j*n m*	Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with 1.
-o *list*	Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
-t*c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

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

The conventions of *join, sort, comm, uniq* and *awk* (1) are wildly incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

NAME

locate - identify a UNIX system command using keywords

SYNOPSIS

[help] locate
[help] locate [keyword1 [keyword2] ...]

DESCRIPTION

The *locate* command is part of the UNIX system Help Facility, and provides on-line assistance with identifying UNIX system commands.

Without arguments, the initial *locate* screen is displayed from which the user may enter keywords functionally related to the action of the desired UNIX system commands they wish to have identified. A user may enter keywords and receive a list of UNIX system commands whose functional attributes match those in the keyword list, or may exit to the shell by typing q (for "quit"). For example, if you wish to print the contents of a file, enter the keywords "print" and "file". The *locate* command would then print the names of all commands related to these keywords.

Keywords may also be entered directly from the shell, as shown above. In this case, the initial screen is not displayed, and the resulting command list is printed.

More detailed information on a command in the list produced by *locate* can be obtained by accessing the *usage* module of the UNIX system Help Facility. Access is made by entering the appropriate menu choice after the command list is displayed.

From any screen in the Help Facility, a user may execute a command via the shell (*sh* (1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable SCROLL must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your **.profile** file (see *profile* (4)): `''export SCROLL ; SCROLL=no''`. If you later decide that scrolling is desired, SCROLL must be set to **yes** .

Information on each of the Help Facility commands (*starter, locate, usage, glossary,* and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), sh(1), starter(1), usage(1).

term(5) in the *Programmer's Reference Manual.*

WARNINGS

If the shell variable **TERM** (see *sh* (1)) is not set in the user's **.profile** file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term* (5).

NAME

mailx - interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of *mailx* will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see "MBOX" ENVIRONEMENT VARIABLES) for a description of this file). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **-f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary *maillbox*. This gives rise within these pages to the notion of a current *maillbox*.

On the command line, *options* start with a dash (-) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *maillbox*. Command line options are:

-e	Test for presence of mail. *mailx* prints nothing and exits with a successful return code if there is mail to read.
-f [*filename*]	Read messages from *filename* instead of *maillbox*. If no *filename* is specified, the *mbox* is used.
-F	Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONEMENT VARIABLES).
-h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. (See **addsopt** under ENVIRONEMENT VARIABLES)
-H	Print header summary only.
-i	Ignore interrupts. See also "ignore" (ENVIRONEMENT VARIABLES).

-n	Do not initialize from the system default *mailx.rc* file.
-N	Do not print initial header summary.
-r *address*	Pass *address* to network delivery software. All tilde commands are disabled. (See **addsopt** under ENVIRONEMENT VARIABLES)
-s *subject*	Set the Subject header field to *subject*.
-u *user*	Read *user 's maillbox*. This is only effective if *user 's maillbox* is not read protected.
-U	Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable. (See **addsopt** under ENVIRONEMENT VARIABLES)

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A "subject" longer than 1024 characters will cause *mailx* to dump core) As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **set** and **unset** commands. See ENVIRONEMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to to undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp* (1) for recording outgoing mail on paper. Alias groups are set by the **alias** command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

```
[ command ] [ msglist ] [ arguments ]
```

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

n	Message number **n.**
.	The current message.
^	The first undeleted message.
$	The last message.
*****	All messages.
n-m	An inclusive range of message numbers.
user	All messages from **user.**
/string	All messages with **string** in the subject line (case ignored).
:c	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh* (1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* tries to execute commands from the optional system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **Copy**, **edit**, **followup**, **Followup**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, and **visual**. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional, and must be constructed locally.

COMMANDS

The following is a complete list of *mailx* commands:

! *shell-command*

Escape to the shell. See SHELL (ENVIRONEMENT VARIABLES).

*comment*

Null command (comment). This may be useful in *.mailrc* files.

=

Print the current message number.

?

Prints a summary of commands.

alias *alias name ...*
group *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates *name ...*

Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alternates** prints the current list of alternate names. See also "allnet" (ENVIRONEMENT VARIABLES).

cd [*directory*]
chdir [*directory*]

Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*

Copy messages to the file without marking the messages as saved. Otherwise equivalent to the **save** command.

Copy [*msglist*]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the **Save** command.

delete [*msglist*]

Delete messages from the *maillbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONEMENT VARIABLES).

discard [*header-field ...*]
ignore [*header-field ...*]

Suppresses printing of the specified header fields when displayihg messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **Print** and **Type** commands override this command.

dp [*msglist*]
dt [*msglist*]

Delete the specified messages from the *maillbox* and print the next message after the last one deleted. Roughly equivalent to a **delete** command followed by a **print** command.

echo *string ...*

Echo the given strings (like *echo* (1)).

edit [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the **EDITOR** variable is used to get the name of the editor (see ENVIRONEMENT VARIABLES). Default editor is *ed* (1).

exit
xit

xit from *mailx*, without changing the *maillbox*. No messages are saved in the *mbox* (see also **q**it).

file [*filename*]
folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current *maillbox*.
% user	the *maillbox* for **user.**
#	the previous file.
&	the current *mbox*.

Default file is the current *maillbox*.

folders

Print the names of the files in the directory set by the "folder" variable (see ENVIRONEMENT VARIABLES).

followup [*message*]

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **F**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONEMENT VARIABLES).

Followup [*msglist*]

Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **f**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONEMENT VARIABLES).

from [*msglist*]

Prints the header summary for the specified messages.

group alias *name* ...
alias alias *name* ...

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]

Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONEMENT VARIABLES). See also the **z** command.

help

Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]

Holds the specified messages in the *maillbox*.

if *s* | *r*
mail-commands
else
mail-commands
endif

Conditional execution, where *s* will execute following *mail-commands*, up to an **else** or **endif**, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore *header-field ...*
discard *header-field ...*

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **Print** and **Type** commands override this command.

list

Prints all commands available. No explanation is given.

mail *name ...*

Mail a message to the specified users.

Mail *name*

Mail a message to the specified user and record a copy of it in a file named after that user.

mbox [*msglist*]

Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See **MBOX** (ENVIRONEMENT VARIABLES) for a description of this file. See also the **exit** and **quit** commands.

next [*message*]

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONEMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]

Preserve the specified messages in the *maillbox*.

Print [*msglist*]
Type [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** command.

print [*msglist*]
type [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the **PAGER** variable. The default command is *pg* (1) (see ENVIRONEMENT VARIABLES).

quit

Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *maillbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]

Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONEMENT VARIABLES).

reply [*message*]
respond [*message*]

Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONEMENT VARIABLES).

Save [*msglist*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and "outfolder" (ENVIRONEMENT VARIABLES).

save [*filename*]
save [*msglist*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *maillbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONEMENT VARIABLES and the **exit** and **quit** commands).

set
set *name*
set *name=string*
set *name=number*

Define a variable called *name*. The variable may be given a null, string, or numeric value. **Set** by itself prints all defined variables and their values. See ENVIRONEMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell

Invoke an interactive shell (see also **SHELL** (ENVIRONEMENT VARIABLES)).

size [*msglist*]

Print the size in characters of the specified messages.

source *filename*

Read commands from the given file and return to command mode.

top [*msglist*]

Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONEMENT VARIABLES). The default is 5.

touch [*msglist*]

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See **exit** and **quit**.

Type [*msglist*]
Print [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** command.

type [*msglist*]
print [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the **PAGER** variable. The default command is *pg* (1) (see ENVIRONEMENT VARIABLES).

undelete [*msglist*]

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONEMENT VARIABLES).

unset *name ...*

Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version

Prints the current version and release date.

visual [*msglist*]

Edit the given messages with a screen editor. The messages are placed in a temporary file and the **VISUAL** variable is used to get the name of the editor (see ENVIRONEMENT VARIABLES).

write [*msglist*] *filename*

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the save command.

xit
exit

Exit from *mailx*, without changing the *maillbox*. No messages are saved in the *mbox* (see also quit).

z [+ | -]

Scroll the header display forward or backward one screen-full. The number of headers displayed is set by the "screen" variable (see ENVIRONEMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONEMENT VARIABLES) for changing this special character.

~! *shell-command*

Escape to the shell.

~.

Simulate end of file (terminate message input).

~: *mail-command*
~_ *mail-command*

Perform the command-level request. Valid only when sending a message while reading mail.

~?

Print a summary of tilde escapes.

~A

Insert the autograph string "Sign" into the message (see ENVIRONEMENT VARIABLES).

~a

Insert the autograph string "sign" into the message (see ENVIRONEMENT VARIABLES).

~b *name* ...

Add the *names* to the blind carbon copy (Bcc) list.

~c *name* ...

Add the *names* to the carbon copy (Cc) list.

~d

Read in the *dead-letter* file. See **DEAD** (ENVIRONEMENT VARIABLES) for a description of this file.

~e

Invoke the editor on the partial message. See also **EDITOR** (ENVIRONEMENT VARIABLES).

~f [*msglist*]

Forward the specified messages. The messages are inserted into the message, without alteration.

~h

Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *string*

Insert the value of the named variable into the text of the message. For example, **~A** is equivalent to **'~i \ Sign.'**

~m [*msglist*]

Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p

Print the message being entered.

~q

Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead-letter*. See **DEAD** (ENVIRONEMENT VARIABLES) for a description of this file.

~r *filename*
~~< \ *filename*
~~< \ !*shell-command*

Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string* ...

Set the subject line to *string*.

~t *name* ...

Add the given *names* to the To list.

~v

Invoke a preferred screen editor on the partial message. See also **VISUAL** (ENVIRONEMENT VARIABLES).

~w *filename*

Write the partial message onto the given file, without the header.

~x

Exit as with **~q** except the message is not saved in *dead-letter*.

~| *shell-command*

Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONEMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*	The user's base of operations.
MAILRC=*filename*	The name of the start-up file. Default is **$HOME/.mailrc.**

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **unset** command may be used to erase variables.

addsopt	Enabled by default. If */bin/mail* is not being used as the deliverer, **noaddsopt** should be specified. (See WARNINGS below)
allnet	All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet.** See also the **alternates** command and the "metoo" variable.
append	Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**
askcc	Prompt for the Cc list after message is entered. Default is **noaskcc.**
asksub	Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.
autoprint	Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint.**
bang	Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi* (1). Default is **nobang.**

cmd=*shell-command*

Set the default command for the **pipe** command. No default value.

conv=*conversion* Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **-U** command line option.

crt=*number* Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (*pg* (1) by default). Disabled by default.

DEAD=*filename* The name of the file in which to save partial letters in case of untimely interrupt. Default is **$HOME/dead.letter.**

debug Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug.**

dot Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot.**

EDITOR=*shell-command*

The command to run when the **edit** or **~e** command is used. Default is *ed* (1).

escape=*c* Substitute *c* for the ~ escape character. Takes effect with next message sent.

folder=*directory* The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), **$HOME** is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header Enable printing of the header summary when entering *mailx*. Enabled by default.

hold Preserve all messages that are read in the *maillbox* instead of putting them in the standard *mbox* save file. Default is **nohold.**

ignore Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore.**

ignoreeof Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the ~. command. Default is **noignoreeof.** See also "dot" above.

keep When the *maillbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave Keep messages that have been saved in other files in the *maillbox* instead of deleting them. Default is **nokeepsave.**

MBOX=*filename* The name of the file to save messages which have been read. The **xit** command overrides this function, as does saving the message explicitly in another file. Default is **$HOME/mbox.**

metoo If your login appears as a recipient, do not delete it from the list. Default is **nometoo.**

LISTER= *shell-command*

The command (and options) to use when listing the contents of the "folder" directory. The default is *ls* (1).

onehop When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder.** See "folder" above and the **S**ave, **C**opy, **f**ollowup, and **F**ollowup commands.

page Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage.**

PAGER=*shell-command*

The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg* (1).

prompt=*string* Set the *command mode* prompt to *string*. Default is "? ".

quiet Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet.**

record=*filename* Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save Enable saving of messages in *dead-letter* on interrupt or delivery error. See **DEAD** for a description of this file. Enabled by default.

screen *number* Sets the number of lines in a screen-full of headers for the **headers** command.

sendmail=*shell-command*

Alternate command for delivering messages. Default is *mail* (1).

sendwait Wait for background mailer to finish before returning. Default is **nosendwait.**

SHELL =*shell-command*

The name of a preferred command interpreter. Default is *sh* (1).

showto When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string* The variable inserted into the text of a message when the **~a** (autograph) command is given. No default (see also **~i** (TILDE ESCAPES)).

Sign=*string* The variable inserted into the text of a message when the **~A** command is given. No default (see also **~i** (TILDE ESCAPES)).

toplines=*number* The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*

The name of a preferred screen editor. Default is *vi* (1).

FILES

/usr/lib/mailx/mailx.help	
$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	optional global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

ls(1), mail(1), pg(1).

WARNINGS

The **-h, -r** and **-U** options can be used only if *mailx* is built with a delivery program other than */bin/mail.*

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset.**

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail* (1) (the standard mail delivery program).

MAILX(1)

NAME

makekey - generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

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

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

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

makekey is intended for programs that perform encryption. Usually, its input and output will be pipes.

SEE ALSO

ed(1), crypt(1), vi(1).

passwd(4) in the *Programmer's Reference Manual.*

MAKEKEY(1)

NAME

man - print entries in this manual

SYNOPSIS

man [options] [section] titles

DESCRIPTION

man locates and prints the entry of this manual named title in the specified section. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The title is entered in lower case. The section number may not have a letter suffix. If no section is specified, the whole manual is searched for title and all occurrences of it are printed. Options and their meanings are:

-T*term*	Print the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type help term2. The default value of term is 450.
-w	Print on the standard output only the path names of the entries, relative to **/usr/catman**, or to the current directory for **-d** option.
-d	Search the current directory rather than **/usr/catman**; requires the full file name (e.g., cu.1c, rather than just cu).
-c	Causes man to invoke *col* (1); note that *col* (1) is invoked automatically by man unless term is one of 300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620, and X.

man examines the environment variable **$TERM** [see *environ* (5)] and attempts to select options that adapt the output to the terminal being used. The **-T*term*** option overrides the value of **$TERM**; in particular, one should use **-Tlp** when sending the output of man to a line printer.

Section may be changed before each title.

As an example:

```
man man
```

would reproduce on the terminal this entry, as well as any other entries named man that may exist in other sections of the manual.

FILES

/usr/catman/?_man/man[1-8]/* Preformatted manual entries

SEE ALSO

term(5) in the Programmer's Reference Manual.

CAVEAT

The man command prints manual entries that were formatted by *nroff* when the UNIX system was installed. Entries are originally formatted with terminal type 37, and are printed using the correct terminal filters as

derived from the -T*term* and $TERM settings. Typesetting or other non-standard printing of manual entries requires installation of the system ***Documenter's Workbench***.

NAME

nawk - pattern scanning and processing language

SYNOPSIS

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

DESCRIPTION

nawk is a new version of *awk* that provides capabilities unavailable in previous versions. This version will become the default version of *awk* in the next major UNIX system release.

OPTIONS

-F *re* defines the input field separator to be the regular expression *re*. An input line is normally made up of fields separated by white space. The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

This white-space-default can also be changed by using the FS built-in variable. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

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

Parameters, in the form x=... y=... may be passed to *nawk*, where x and y are *nawk* built-in variables (see list below).
Input files are read in order; if there are no files, the standard input is read.The file name - means the standard input. Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

A pattern-action statement has the form:

pattern { action }

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

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

expression relop expression

expression matchop regular expression

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

var **in** array,

or a Boolean combination of these.

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

Regular expressions are as in *egrep* [see *grep*(1)]. In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second pattern.

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

Other built-in variables include:

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

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

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

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

The **print** statement prints its arguments on the standard output, or on a file if *>expression* is present, or on a pipe if | *cmd* is present. The arguments are separated by the current output field separator and terminated by the output record separator. The **printf** statement formats its expression list according to the format [see *printf*(3S) in the *Programmer's Reference Manual*].

nawk has a variety of built-in functions:

arithmetic, string, input/output, and **general.**

The **arithmetic** functions are:

atan2, cos, exp, int, log, rand, srand, sin, sqrt.

int	truncates its argument to an integer.
rand	returns a random number between 0 and 1.
srand (expr)	sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The **string** functions are:

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

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

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

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

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

sprintf(fmt, expr, expr, ...)
formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

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

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

The **input/output** and **general** functions are:

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

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

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

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

getline var sets variable *var* instead.

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

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

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

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

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

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

EXAMPLES

print lines longer than 72 characters: **length > 72**

Print first two fields in opposite order: **{ print $2, $1 }**

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

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

Add up first column, print sum and average:

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

Print fields in reverse order: **{ for (i = NF; i > 0; --i) print $i }**

Print all lines between start/stop pairs: **/start/, /stop/**

Print all lines whose first field is different from previous one: **$1 != prev { print; prev = $1 }**

Simulate *echo*(1):

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

Print file, filling
in page numbers
starting at 5:

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

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

SEE ALSO

grep(1), *sed(1)*.
lex(1), *printf(3S)* in the Programmer's Reference Manual.
Programmer's Guide.

BUGS

Input white space is not preserved on output if fields are involved. There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string, concatenate the null string ("") to it.

NAME

newform - change the format of a text file

SYNOPSIS

newform [-s] [-i*tabspec***] [-o***tabspec***] [-b***n***] [-e***n***] [-p***n***] [-a***n***] [-f] [-c***char***] [-l***n***] [files]**

DESCRIPTION

newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **-s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like **" -e 15 -l 60"** will yield results different from **" -l 60 -e 15"**. Options are applied to all *files* on the command line.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform -s -i -l -a -e file-name
```

-i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *tabspec* recognizes all tab specification forms described in *tabs* (1). In addition, *tabspec* may be **--**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec* (4)). If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** expects no tabs; if any are found, they are treated as **-1**.

-o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **-i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0**

means that no spaces will be converted to tabs on output.

-b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **-b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -l1 -b7 file-name
```

-e*n* Same as **-b***n* except that characters are truncated from the end of the line.

-p*n* Prefix *n* characters (see **-c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

-a*n* Same as **-p***n* except characters are appended to the end of a line.

-f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **-o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.

-c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

-l*n* Set the effective line length to *n* characters. If *n* is not entered, **-l** defaults to 72. The default line length without the **-l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **-i** to expand tabs to spaces).

The **-l1** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: \ ... *newform* was called with a bad option.

not -s format There was no tab on one line.

can't open file Self-explanatory.

internal line too long
A line exceeds 512 characters after being expanded in the internal work buffer.

tabspec in error A tab specification is incorrectly formatted, or specified tab stops are not ascending.

tabspec indirection illegal

A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0 - normal execution

1 - for any error

SEE ALSO

csplit(1), tabs(1).

fspec(4) in the *Programmer's Reference Manual.*

BUGS

newform normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was **-o--** , and was preceded by either a **-o--** or a **-i--** , the tab specification format line will be incorrect.

NEWFORM(1)

NAME

news - print news items

SYNOPSIS

news [**-a**] [**-n**] [**-s**] [items]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news.**

When invoked without arguments, *news* prints the contents of all current files in **/usr/news** , most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

-a option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

-n option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

-s option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile.**

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5) in the *Programmer's Reference Manual.*

NEWS(1)

NAME

nl - line numbering filter

SYNOPSIS

nl [-h*type***] [-b***type***] [-f***type***] [-v***start#***] [-i***incr***] [-p] [-l***num***] [-s***sep***] [-w***width***] [-n***format***] [-d***delim***]** file

DESCRIPTION

nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	***Start of***
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

-b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:

-h*type* Same as **-b*type*** except for header. Default *type* for logical page header is **n** (no lines numbered).

- **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).

-f*type* Same as **-b*type*** except for footer. Default for logical page footer is **n** (no lines numbered).

-v*start#* *start#* is the initial value used to number logical page lines. Default is **1**.

-i*incr* *incr* is the increment value used to number logical page lines. Default is **1**.

-p	Do not restart numbering at logical page delimiters.
-l*num*	*num* is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha** , **-ba** , and/or **-fa** option is set). Default is **1.**
-s*sep*	*sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
-w*width*	*width* is the number of characters to be used for the line number. Default *width* is **6.**
-n*format*	*format* is the line numbering format. Recognized values are: **ln** , left justified, leading zeroes suppressed; **rn** , right justified, leading zeroes supressed; **rz** , right justified, leading zeroes kept. Default *format* is **rn** (right justified).
-d*xx*	The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **-d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nroff - text formatting

SYNOPSIS

nroff [option] ... [file] ...

DESCRIPTION

nroff formats text in the named *files* for typewriter-like devices. The full capabilities of *nroff* are described in the *Nroff/Troff User's Manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input.

The options, which may appear in any order so long as they appear *before* the files, are:

-o*list*	Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N* - *M* means pages *N* through *M;* an initial -*N* means from the beginning to page *N;* and a final *N* - means from *N* to the end.
-n*N*	Number first generated page *N*.
-s*N*	Stop every *N* pages. *nroff* will halt prior to every *N* pages (default *N =1)* to allow paper loading or changing, and will resume upon receipt of a newline.
-m*name*	Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.
-r*aN*	Set register *a* (one-character) to *N*.
-i	Read standard input after the input files are exhausted.
-q	Invoke the simultaneous input-output mode of the **rd** request.
-T*name*	Prepare output for specified terminal. Known *names* are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).
-e	Produce equally-spaced words in adjusted lines, using full terminal resolution.
-h	Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

FILES

/usr/lib/tmac/tmac.*	
/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

J. F. Ossanna, *Nroff/Troff user's manual*

B. W. Kernighan, *A TROFF Tutorial*

troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

NAME

pack, pcat, unpack - compress and expand files

SYNOPSIS

pack [-] [**-f**] name ...
pcat name ...
unpack name ...

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.***z** with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

- the file appears to be already packed;
- the file name has more than 12 characters;
- the file has links;
- the file is a directory;
- the file cannot be opened;
- no disk storage blocks will be saved by packing;
- a file called *name.***z** already exists;
- the **.z** file cannot be created;

- an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

pcat does for packed files what *cat* (1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name.z* use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

```
pcat name >nnn
```

pcat returns the number of files it was unable to unpack. Failure may occur if:

- the file name (exclusive of the **.z**) has more than 12 characters;
- the file cannot be opened;
- the file does not appear to be the output of *pack*.

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat* , as well as for the following:

- a file with the "unpacked" name already exists;
- if the unpacked file cannot be created.

SEE ALSO

cat(1).

PASTE(1)

NAME

paste - merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste -d list file1 file2 ...
paste -s [**-d** list] file1 file2 ...

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1, file2,* etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat* (1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list.* Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if - is used in place of a file name.

The meanings of the options are:

-d — Without this option, the new-line characters of each but the last file (or last line in case of the **-s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list — One or more characters immediately following **-d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **-s** option), the lines from the last file are always terminated with a new-line character, not from the *list.* The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), **** (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use **-d**"\\\\").

-s — Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **-d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

- — May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

```
ls | paste -d" " -
```

list directory in one column

```
ls | paste - - - -
```

list directory in four columns

```
paste -s -d"\t\n" file
```

combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long

Output lines are restricted to 511 characters.

too many files

Except for **-s** option, no more than 12 input files may be specified.

NAME

pg - file perusal filter for CRTs

SYNOPSIS

pg [*-number*] [**-p** *string*] [**-cefns**] [*+linenumber*] [*+/ pattern /*] [**files ...**]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a CRT. (The file name - and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo* (4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

-number	An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).
-p ***string***	Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is " **:** ".
-c	Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo* (4) data base.
-e	Causes *pg not* to pause at the end of each file.
-f	Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The **-f** option inhibits *pg* from splitting lines.
-n	Normally, commands must be terminated by a *newline* character. This option causes an automatic end of command as soon as a command letter is entered.
-s	Causes *pg* to print all messages and prompts in standout mode (usually inverse video).
+linenumber	Start up at *linenumber*.

+/*pattern*/ Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1) *newline* **or** *blank*

This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d or ^D** Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or ^L Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed* (1) are available. They must always be terminated by a *newline*, even if the **-n** option is specified.

i/*pattern*/ Search forward for the *i*th (default *i* =1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i ^*pattern*^
i?*pattern*?

Search backwards for the *i*th (default *i* =1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command

to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n**	Begin perusing the *i th* next file in the command line. The *i* is an unsigned number, default value is 1.
*i***p**	Begin perusing the *i th* previous file in the command line. *i* is an unsigned number, default is 1.
*i***w**	Display another window of text. If *i* is present, set the window size to *i*.
s ***filename***	Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a *newline,* even if the **-n** option is specified.
h	Help by displaying an abbreviated summary of available commands.
q or Q	Quit *pg*.
!*command*	*Command* is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a *newline,* even if the **-n** option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat* (1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg's* current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/*	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1).

terminfo(4) in the *Programmer's Reference Manual.*

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

NAME

ptx - permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *ptx* produces output in the form:

```
.xx "tail" "before keyword" "keyword and after" "head"
```

where .xx may be an *nroff* or *troff* (1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *tail* and *head,* at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

-f	Fold upper and lower case letters for sorting.
-t	Prepare the output for the phototypesetter; the default line length is 100 characters.
-w*n*	Use the next argument, *n,* as the width of the output line. The default line length is 72 characters.
-g*n*	Use the next argument, *n,* as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.
-o*only*	Use as keywords only the words given in the *only* file.
-i*ignore*	Do not use as keywords any words given in the *ignore* file. If the **-i** and **-o** options are missing, use **/usr/lib/eign** as the *ignore* file.
-b*break*	Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.
-r	Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx.*

FILES

/usr/bin/sort

/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

sar - system activity reporter

SYNOPSIS

sar [-ubdycwaqvmprDSA] [-o *file***] t [** *n* **]**
sar [-ubdycwaqvmprDSA] [-s *time***] [-e** *time***] [-i** *sec***] [-f** *file***]**

DESCRIPTION

sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If the **-o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded *file*, either the one specified by **-f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **-s** and **-e** *time* arguments of the form *hh [: mm [: ss]]*. The **-i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

-u Report CPU utilization (the default):
%usr, %sys, %wio, %idle - portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle. When used with **-D**, %sys is split into percent of time servicing requests from remote machines (%sys remote) and all other system time (%sys local).

-b Report buffer activity:
bread/s, bwrit/s - transfers per second of data between system buffers and disk or other block devices;

lread/s, lwrit/s - accesses of system buffers;

%rcache, %wcache - cache hit ratios, i. e., (1-bread/lread) as a percentage;

pread/s, pwrit/s - transfers via raw (physical) device mechanism.

-d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:

%busy, avque - portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;

r+w/s, blks/s - number of data transfers from or to device, number of bytes transferred in 512-byte units;

avwait, avserv - average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

-y Report TTY device activity:
rawch/s, canch/s, outch/s - input character rate, input character rate processed by canon, output character rate;

rcvin/s, xmtin/s, mdmin/s - receive, transmit and modem interrupt rates.

-c Report system calls:
scall/s - system calls of all types;

sread/s, swrit/s, fork/s, exec/s - specific system calls;

rchar/s, wchar/s - characters transferred by read and write system calls. When used with **-D**, the system calls are split into incoming, outgoing, and strictly local calls.

-w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s - number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);

pswch/s - process switches.

-a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

-q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc - run queue of processes in memory and runnable;

swpq-sz, %swpocc - swap queue of processes swapped out but ready to run.

-v Report status of process, i-node, file tables:
text-sz, proc-sz, inod-sz, file-sz, lock-sz - entries/size for each table, evaluated once at sampling point;

ov - overflows that occur between sampling points for each table.

-m Report message and semaphore activities:
msg/s, sema/s - primitives per second.

-p Report paging activities:
vflt/s - address translation page faults (valid page not in memory);

pflt/s - page faults from protection errors (illegal access to page) or "copy-on-writes";

pgfil/s - vflt/s satisfied by page-in from file system;

rclm/s - valid pages reclaimed for free list.

-r Report unused memory pages and disk blocks:
freemem - average pages available to user processes;

freeswap - disk blocks available for process swapping.

-D Report Remote File Sharing activity:
When used in combination with **-u** or **-c**, it causes *sar* to produce the remote file sharing version of the corresponding report. **-u** is assumed when neither **-u** or **-c** is specified.

-S Report server and request queue status:
Average number of Remote File Sharing servers on the system (serv/lo-hi), % of time receive descriptors are on the request queue (request %busy), average number of receive descriptors waiting for service when queue is occupied (request avg lgth), % of time there are idle servers (server %avail), average number of idle servers when idle ones exist (server avg avail).

-A Report all data. Equivalent to **-udqbwcayvmprSD.**

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar -o temp 60 10
```

To later review disk and tape activity from that period:

```
sar -d -f temp
```

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G).

sar(1M) in the *Administrator's Reference Manual.*

SAR(1)

NAME

sdiff - side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

sdiff uses the output of *diff* (1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x     |     y
a           a
b     <
c     <
d           d
      >     c
```

The following options exist:

-w *n* Use the next argument, *n* , as the width of the output line. The default line length is 130 characters.

-l Only print the left side of any lines that are identical.

-s Do not print identical lines.

-o *output* Use the next argument, *output* , as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff* (1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a **%** and waits for one of the following user-typed commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

setpgrp - run a program with a new process group

SYNOPSIS

setpgrp *command* [*arguments*]

DESCRIPTION

The command *setpgrp* executes "command" with the new process group id.

SEE ALSO

getpgrp(2)

SETPGRP(1)

NAME

sno - SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

sno is a SNOBOL compiler and interpreter (with slight differences). *sno* obtains input from the concatenation of the named files and the standard input. All input through a statement containing the label end is considered program and is compiled. The rest is available to syspit.

sno differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

```
a ** b              unanchored search for b.
a *x* b = x c       unanchored assignment
```

There is no back referencing.

```
x = "abc"
a *x* x       is an unanchored search for abc.
```

Function declaration is done at compile time by the use of the (non-unique) label define. Execution of a function call begins at the statement following the define. Functions cannot be defined at run time, and the use of the name define is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except define (even end) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on end cannot merely name a label.

If start is a label in the program, program execution will start there. If not, execution begins with the first executable statement; define is not an executable statement.

There are no built-in functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and * must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable sysppt is not available.

SEE ALSO

awk(1).

NAME

spell, hashmake, spellin, hashcheck - find spelling errors

SYNOPSIS

spell [**-v**] [**-b**] [**-x**] [**-l**] [+ local_file] [files]
/usr/lib/spell/hashmake
/usr/lib/spell/spellin n
/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most *troff* (1), *tbl* (1), and *eqn* (1) constructions.

Under the **-v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **-b** option, British spelling is checked. Besides preferring *centre, colour, programme, speciality, travelled,* etc., this option insists upon *-ise* in words like *standardise,* Fowler and the OED to the contrary notwithstanding.

Under the **-x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff* (1)) follows chains of included files (**.so** and **.nx** *troff* (1) requests), *unless* the names of such included files begin with **/usr/lib.** Under the **-l** option, *spell* will follow the chains of *all* included files.

Under the *+local_file* option, words found in *local_file* are removed from *spell 's* output. *local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell 's* own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy-y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell :*

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), sed(1), sort(1), tee(1).

eqn(1), tbl(1), troff(1) in the *DOCUMENTER'S WORKBENCH Software 2.0 Technical Discussion and Reference Manual*.

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

NAME

spline - interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output has two continuous derivatives, and sufficiently many points to look smooth when plotted.

The following *options* are recognized, each as a separate argument:

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

-k The constant k used in the boundary value computation:

$y_0'' = ky_1'', y_n'' = ky''_{n-1}$

is set by the next argument (default $k = 0$).

-n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).

-p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.

-x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

NAME

split - split a file into pieces

SYNOPSIS

split [*-n*] [file [name]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if **-** is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

starter - information about the UNIX system for beginning users

SYNOPSIS

[help] starter

DESCRIPTION

The UNIX system Help Facility command *starter* provides five categories of information about the UNIX system to assist new users.

The five categories are:

- commands a new user should learn first
- UNIX system documents important for beginners
- education centers offering UNIX system courses
- ocal environment information
- on-line teaching aids installed on the UNIX system

The user may choose one of the above categories by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit"). When a category is chosen, the user will receive one or more pages of information pertaining to it.

From any screen in the Help Facility, a user may execute a command via the shell (*sh* (1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your **.profile** file (see *profile* (4)): `''export SCROLL ; SCROLL=no''`. If you later decide that scrolling is desired, SCROLL must be set to **yes**.

Information on each of the Help Facility commands (*starter, locate, usage, glossary,* and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), locate(1), sh(1), usage(1).

term(5) in the *Programmer's Reference Manual.*

WARNINGS

If the shell variable **TERM** (see *sh* (1)) is not set in the user's **.profile** file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term* (5).

STARTER(1)

NAME

sum - print checksum and block count of a file

SYNOPSIS

sum [**-r**] file

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

tabs - set tabs on a terminal

SYNOPSIS

tabs [tabspec] [-Ttype] [+m n]

DESCRIPTION

tabs sets the tab stops on the user's terminal according to the tab specification *tabspec,* after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec — Four types of tab specification are accepted for *tabspec.* They are described below: canned (*-code*), repetitive (*-n*), arbitrary (*n1,n2,...*), and file (*--file*). If no *tabspec* is given, the default value is **-8** , i.e., UNIX system "standard" tabs. The lowest column number is 1. Note that for *tabs* , column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

- code — Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

-a 1,10,16,36,72

Assembler, IBM S/370, first format

-a2 1,10,16,40,72

Assembler, IBM S/370, second format

-c 1,8,12,16,20,55

COBOL, normal format

-c2 1,6,10,14,49

COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see *fspec* (4)):

<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67

COBOL compact format (columns 1-6 omitted), with more tabs than **-c2.** This is the recommended format for COBOL. The appropriate format specification is (see *fspec* (4)):

<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23

FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61

PL/I

-s 1,10,55

SNOBOL

-u 1,12,20,44

UNIVAC 1100 Assembler

-n A *repetitive* specification requests tabs at columns 1+ *n*, 1+2* *n* , etc. Of particular importance is the value **8** : this represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0** , implying no tabs at all.

n1 , *n2* ,... The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1,10,20,30**, and **1,10,+10,+10** are considered identical.

-- *file* If the name of a *file* is given, *tabs* reads the first line of the file, searching for a format specification (see *fspec* (4)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr* (1) command:

tabs -- file; **pr** file

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

-T*type* *tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term* (5). If no **-T** flag is supplied, *tabs* uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* (see *environ* (5)), *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n* , the value assumed is **10**. For a TermiNet, the first value in the tab list should be **1** , or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

```
tabs -a
```

example using *-code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

```
tabs -8
```

example of using *-n* (*repetitive* specification), where *n* is **8** , causes tabs to be set every eighth position:

1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

```
tabs 1,8,36
```

example of using *n1* , *n2* ,... (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

```
tabs --$HOME/fspec.list/att4425
```

example of using *--file* (*file* specification) to indicate that tabs should be set according to the first line of **$HOME/fspec.list/att4425** (see *fspec* (4)).

DIAGNOSTICS

illegal tabs

when arbitrary tabs are ordered incorrectly

illegal increment

when a zero or missing increment is found in an arbitrary specification

unknown tab code

when a *canned* code cannot be found

can't open

if *--file* option used, and file can't be opened

file indirection

if *--file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted

SEE ALSO

newform(1), pr(1), tput(1).

fspec(4), terminfo(4), environ(5), term(5) in the *Programmer's Reference Manual.*

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The *tabspec* used with the *tabs* command is different from the one used with the *newform* (1) command. For example, **tabs -8** sets every eighth position; whereas **newform -i-8** indicates that tabs are set every eighth position.

NAME

tail - deliver the last part of a file

SYNOPSIS

tail [(± [**number**][**lbc** [**f**]]] [file]

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance ***+number*** from the beginning, or ***-number*** from the end of the input (if *number* is null, the value 10 is assumed). *number* is counted in units of lines, blocks, or characters, according to the appended option **l** , **b** , or **c**. When no units are specified, counting is by lines.

With the **-f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f fred
```

will print the last ten lines of the file **fred** , followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail -15cf fred
```

will print the last 15 characters of the file **fred** , followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1M) in the *D-NIX 5.3 Reference Manual.*

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

WARNING

The *tail* command will only tail the last 4096 bytes of a file regardless of its line count.

TAIL(1)

NAME

tbl - format tables for nroff or troff

SYNOPSIS

tbl [files] ...

DESCRIPTION

tbl is a preprocessor for formatting tables for *nroff* or *troff* (1). The input files are copied to the standard output, except for lines between **.TS** and **.TE** command lines, which are assumed to describe tables and are reformatted. Details are given in the *tbl* (1) reference manual.

EXAMPLE

As an example, letting -> represent a tab (which should be typed as a genuine tab) the input

```
.TS
center box ;
cB s s
cI | cI s
^ | n n .

Household Population

Town->Households
->Number->Size
Bedminster->789->3.26
Bernards Twp.->3087->3.74
Bernardsville->2018->3.30
Bound Brook->3425->3.04
Branchburg->1644->3.49
Bridgewater->7897->3.81
Far Hills->240->3.19
.TE
```

yields

Household Population		
	Households	
Town	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

If no arguments are given, *tbl* reads the standard input, so it may be used as a filter. When *tbl* is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

SEE ALSO

troff(1), eqn(1)

M. E. Lesk, *TBL*.

NAME

tee - pipe fitting

SYNOPSIS

tee [**-i**] [**-a**] [file] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *files*. The

-i	ignore interrupts;
-a	causes the output to be appended to the *files* rather than overwriting them.

TEE(1)

NAME

tput - initialize a terminal or query terminfo database

SYNOPSIS

tput [-Ttype] capname [parms ...]
tput [-Ttype] **init**
tput [-Ttype] **reset**
tput [-Ttype] **longname**

DESCRIPTION

tput uses the *terminfo* (4) database to make the values of terminal-dependent capabilities and information available to the shell (see *sh* (1)), to initialize or reset the terminal, or return the long name of the requested terminal type. *tput* outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, *tput* simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code ($?, see *sh* (1)) to be sure it is **0**. (See EXIT CODES and DIAGNOSTICS below.) For a complete list of capabilities and the *capname* associated with each, see *terminfo* (4).

-T*type* indicates the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable TERM. If **-T** is specified, then the shell variables LINES and COLUMNS and the layer size (see *layers*(1)) will not be referenced.

capname indicates the attribute from the *terminfo* (4) database.

parms If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.

init If the *terminfo* (4) database is present and an entry for the user's terminal exists (see **-T*type***, above), the following will occur: (1) if present, the terminal's initialization strings will be output **(is1, is2, is3, if, iprog)**, (2) any delays (e.g., newline) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output if present **(rs1, rs2, rs3, rf)**. If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname If the *terminfo* (4) database is present and an entry for the user's terminal exists (see **-T*type*** above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the *terminfo* (4) database (see *term* (5)).

EXAMPLES

```
tput init
```

Initialize the terminal according to the type of terminal in the environmental variable **TERM.** This command should be included in everyone's .profile after the environmental variable **TERM** has been exported, as illustrated on the *profile* (4) manual page.

```
tput -T5620 reset
```

Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable **TERM.**

```
tput cup 0 0
```

Send the sequence to move the cursor to row **0,** column **0** (the upper left corner of the screen, usually known as the "home" cursor position).

```
tput clear
```

Echo the clear-screen sequence for the current terminal.

```
tput cols"
```

Print the number of columns for the current terminal.

```
tput -T450 cols
```

Print the number of columns for the 450 terminal.

```
bold='tput smso'
offbold='tput rmso'
```

Set the shell variables **bold,** to begin stand-out mode sequence, and **offbold,** to end standout mode sequence, for the current terminal. This might be followed by a prompt:

```
echo "${bold}Please type in your name: ${offbold}"
tput hc
```

Set exit code to indicate if the current terminal is a hardcopy terminal.

```
tput cup 23 4
```

Send the sequence to move the cursor to row 23, column 4.

```
tput longname
```

Print the long name from the *terminfo* (4) database for the type of terminal specified in the environmental variable **TERM.**

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/include/curses.h	*curses* (3X) header file
/usr/include/term.h	*terminfo* (4) header file
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of *terminfo* (4)

SEE ALSO

stty (1), tabs (1).

profile(4), terminfo(4) in the *Programmer's Reference Manual.*

EXIT CODES

If *capname* is of type boolean, a value of **0** is set for TRUE and **1** for FALSE.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type.* To determine if *capname* is defined for this terminal *type,* the user must test the value of standard output. A value of **-1** means that *capname* is not defined for this terminal *type.*

Any other exit code indicates an error; see **DIAGNOSTICS,** below.

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

exit code	**error message**
0	-1 (*capname* is a numeric variable that is not specified in the *terminfo* (4) database for this terminal type, e.g. **tput -T450 lines** and **tput -T2621 xmc**)
1	no error message is printed, see **EXIT CODES**, above.
2	usage error
3	unknown terminal *type* or no *terminfo* (4) database
4	unknown *terminfo* (4) capability *capname*

TPUT(1)

NAME

units - conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon.** For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

UNITS(1)

NAME

usage - retrieve a command description and usage examples

SYNOPSIS

[help] usage [**-d**] [**-e**] [**-o**] [command_name]

DESCRIPTION

The UNIX system Help Facility command *usage* retrieves information about UNIX system commands. With no argument, *usage* displays a menu screen prompting the user for the name of a command, or allows the user to retrieve a list of commands supported by *usage*. The user may also exit to the shell by typing q (for "quit).

After a command is selected, the user is asked to choose among a description of the command, examples of typical usage of the command, or descriptions of the command's options. Then, based on the user's request, the appropriate information will be printed.

A command name may also be entered at shell level as an argument to *usage*. To receive information on the command's description, examples, or options, the user may use the **-d** , **-e** , or **-o** options respectively. (The default option is **-d .)**

From any screen in the Help Facility, a user may execute a command via the shell (*sh* (1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your **.profile** file (see *profile* (4)): `''export SCROLL ; SCROLL=no''`. If you later decide that scrolling is desired, **SCROLL** must be set to **yes.**

Information on each of the Help Facility commands (*starter, locate, usage, glossary,* and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), locate(1), sh(1), starter(1).

term(5) in the *Programmer's Reference Manual.*

WARNINGS

If the shell variable **TERM** (see *sh* (1)) is not set in the user's **.profile** file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term* (5).

NAME

vi - screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**-t** *tag*] [**-r** *file*] [**-wn**] [**-R**] [**-x**] [*+command*] *name* ...
view [**-t** *tag*] [**-r** *file*] [**-wn**] [**-R**] [**-x**] [*+command*] *name*
vedit [**-t** *tag*] [**-r** *file*] [**-wn**] [**-R**] [**-x**] [*+command*] *name*

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor *ex* (1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi,* changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

INVOCATION

The following invocation options are interpreted by *vi :*

-t *tag* Edit the file containing the *tag* and position the editor at its definition.

-r *file* Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.

-w*n* Set the default window size to *n*. This is useful when using the editor over a slow speed line.

-R Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.

+command The specified *ex* command is interpreted before editing begins.

-x Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see *crypt* (1)). Also, see the WARNING section at the end of this manual page.

The *name* argument indicates files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

VI MODES

Command Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.

Input Entered by the following options **a i A I o O c C s S R.** Arbitrary text may then be entered. Input mode is nor-

	mally terminated with ESC character, or abnormally with interrupt.
Last line	Reading input for : / ? or !; terminate with CR to execute, interrupt to cancel.

COMMAND SUMMARY

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text*ESC	insert text
abc	
cw*new***ESC**	change word to *new*
ea*s***ESC**	pluralize word
x	delete a character
dw	delete a word
dd	delete a line
3dd	... 3 lines
u	undo previous change
ZZ	exit vi, saving changes
:q!CR	quit, discarding changes
/*text*CR	search for *text*
^U ^D	scroll up or down
:*ex cmd*CR	any *ex* or *ed* command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	
	z G |
scroll amount	^D ^U
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
DEL	(delete or rubout) interrupts
^L	reprint screen if DEL scrambles it
^R	reprint screen if ^L is → key

File manipulation

:wCR	write back changes
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file synonym for **:e #**
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:!*cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	to tag file entry *tag* ^]
:ta,	following word is *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a **CR**.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/pat	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/pat/+n	nth line after *pat*
?*pat***?-***n*	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence

{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z-CR	... at bottom
z.CR	... at center
/*pat*/z-CR	*pat* line at bottom
z*n*.CR	use *n* line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

“	move cursor to previous context
”	... at first non-white in line
m*x*	mark current position with letter *x*
‘*x*	move cursor to mark *x*
’*x*	... at first non-white in line

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
-	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character positioning

^	first non white
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards

^H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	**f** backward
t*x*	upto *x* forward
T*x*	back upto *x*
;	repeat last **f F t** or **T**
,	inverse of **;**
|	to specified column
%	find matching **(** { **)** or **}**

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back **W**
E	to end of **W**

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as **^H**
kill	your kill, erase input this line
****	quotes **^H**, your erase and kill
ESC	ends insertion, back to command
DEL	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before cursor
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command
\=	indent for LISP

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	... before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"*x***p**	put from buffer *x*

"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d*p	retrieve *d*'th last delete

AUTHOR

vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.CORE\term/?/*	subset of compiled terminal description database, supplied on hard disk

SEE ALSO

ed(1), edit(1), ex(1).

WARNING

The **-x** option is provided with the Security Administration Utilities, which is available only in the United States.

Tampering with entries in **/usr/lib/.CORE/term/?/*** or **/usr/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs such as *vi* (1) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

BUGS

Software tabs using **^T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

VI(1)

NAME

vsar - visual system activity reporter

SYNOPSIS

vsar [n]

DESCRIPTION

vsar is a program that continuously shows cpu utilization and how the system is used. The computer status is viewed every five seconds as default.

OPTIONS

n	Time (in seconds) between each system check.

SEE ALSO

sar(1).

NAME

xargs - construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's $PATH. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\ ) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **-i** flag). Flags **-i** , **-l** , and **-n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **-l** vs. **-n**), the last flag has precedence. *flag* values are:

-l*number* *command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **-x** is forced.

-i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **-x** is also forced. {} is assumed for *replstr* if not specified.

-n*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **-x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

-t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

-p Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**-t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

-x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **-x** is forced by the options **-i** and **-l**. When neither of the options **-i** , **-l** , or **-n** are coded, the total length of all arguments must be within the *size* limit.

-s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **-s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-e*eofstr* *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **-e** is not coded. The value **-e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs will terminate if either it receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh* (1)) with an appropriate value to avoid accidentally returning with **-1**.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

1. ls | xargs -p -l ar r arch

2. ls | xargs -p -l | xargs ar r arch

The following will execute *diff* (1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO

sh(1).

XARGS(1)

2

NAME

ci - check in RCS revisions

SYNOPSIS

ci [options] file ...

DESCRIPTION

ci stores new revisions into RCS files. Each file name ending in ,v is taken to be an RCS file, all others are assumed to be working files containing new revisions. *ci* deposits the contents of each working file into the corresponding RCS file.

Pairs of RCS files and working files may be specified in 3 ways (see also the example section of *co* (1B)).

1. Both the RCS file and the working file are given. The RCS file name is of the form **path1/workfile,v** and the working file name is of the form **path2/workfile**, where path1/ and path2/ are (possibly different or empty) paths and workfile is a file name.
2. Only the RCS file is given. Then the working file is assumed to be in the current directory and its name is derived from the name of the RCS file by removing path1/ and the suffix **,v**.
3. Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing path2/ and appending the suffix **,v**.

If the RCS file is omitted or specified without a path, then *ci* looks for the RCS file first in the directory **./RCS** and then in the current directory.

For *ci* to work, the caller's login must be on the access list, except if the access list is empty or the caller is the superuser or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file, unless locking is set to strict (see *rcs* (1B)). A lock held by someone else may be broken with the *rcs* command.

Normally, *ci* checks whether the revision to be deposited is different from the preceding one. If it is not different, *ci* either aborts the deposit (if **-q** is given) or asks whether to abort (if **-q** is omitted). A deposit can be forced with the **-f** option.

For each revision deposited, *ci* prompts for a log message.

The log message should summarize the change and must be terminated with a line containing a single '.' or a **CTRL-D**. If several files are checked in, *ci* asks whether to reuse the previous log message. If the std. input is not a terminal, *ci* suppresses the prompt and uses the same log message for all files. See also **-m**.

The number of the deposited revision can be given by any of the options **-r**, **-f**, **-k**, **-l**, **-u**, or **-q** (see **-r**).

If the RCS file does not exist, *ci* creates it and deposits the contents of the working file as the initial revision (default number: 1.1). The access list is initialized to empty. Instead of the log message, *ci* requests descriptive text (see **-t** below).

OPTIONS

-r[*rev*] assigns the revision number rev to the checked-in revision, releases the corresponding lock, and deletes the working file. This is also the default.

If **rev** is omitted, *ci* derives the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non- tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1. If the caller holds no lock, but he is the owner of the file and locking is not set to strict, then the revision is appended to the trunk.

If **rev** indicates a revision number, it must be higher than the latest one on the branch to which **rev** belongs, or must start a new branch.

If **rev** indicates a branch instead of a revision, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If **rev** indicates a non-existing branch, that branch is created with the initial revision numbered rev.1.

Exception: On the trunk, revisions can be appended to the end, but not inserted.

-f[*rev*] forces a deposit; the new revision is deposited even it is not different from the preceding one.

-k[*rev*] searches the working file for keyword values to determine its revision number, creation date, author, and state (see *co* (1)), and assigns these values to the deposited revision, rather than computing them locally. A revision number given by a command option overrides the number in the working file. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the **-k** option at these sites to preserve its original number, date, author, and state.

-l[*rev*] works like **-r**, except it performs an additional *co* -l for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is use-

ful for saving a revision although one wants to continue editing it after the checkin.

-u[*rev*] works like **-l**, except that the deposited revision is not locked. This is useful if one wants to process (e.g., compile) the revision immediately after checkin.

-q[*rev*] quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless **-f** is given.

-mmsguses the string msg as the log message for all revisions checked in.

-nname assigns the symbolic name name to the number of the checked-in revision. *ci* prints an error message if name is already assigned to another number.

-Nname same as **-n**, except that it overrides a previous assignment of name.

-sstate sets the state of the checked-in revision to the identifier state. The default is Exp.

-t[*txtfile*] writes descriptive text into the RCS file (deletes the existing text). If txtfile is omitted, *ci* prompts the user for text supplied from the std. input, terminated with a line containing a single '.' or control-D. Otherwise, the descriptive text is copied from the file txtfile. During initialization, descriptive text is requested even if **-t** is not given. The prompt is suppressed if std. input is not a terminal.

DIAGNOSTICS

For each revision, *ci* prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status always refers to the last file checked in, and is 0 if the operation was successful, 1 otherwise.

FILE MODES

An RCS file created by *ci* inherits the read and execute permissions from the working file. If the RCS file exists already, *ci* preserves its read and execute permissions. *ci* always turns off all write permissions of RCS files.

FILES

The caller of the command must have read/write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself. A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. *ci* always creates a new RCS file and unlinks the old one. This strategy makes links to RCS files useless.

SEE ALSO

co(1B), ident(1B), rcs(1B), rcsdiff(1B), rcsintro(1B), rcsmerge(1B), rlog(1B), rcsfile(5B), sccstorcs(8B).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

NAME

co - check out RCS revisions

SYNOPSIS

co [options] file ...

DESCRIPTION

co retrieves revisions from RCS files. Each file name ending in **,v** is taken to be an RCS file. All other files are assumed to be working files. *co* retrieves a revision from each RCS file and stores it into the corresponding working file.

Pairs of RCS files and working files may be specified in 3 ways (see also the example section).

1. Both the RCS file and the working file are given. The RCS file name is of the form **path1/workfile,v** and the working file name is of the form **path2/workfile**, where path1/ and path2/ are (possibly different or empty) paths and workfile is a file name.
2. Only the RCS file is given. Then the working file is created in the current directory and its name is derived from the name of the RCS file by removing path1/ and the suffix **,v**.
3. Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing path2/ and appending the suffix **,v**.

If the RCS file is omitted or specified without a path, then co looks for the RCS file first in the directory **./RCS** and then in the current directory.

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later checkin must normally be locked. Locking a revision currently locked by another user fails. (A lock may be broken with the *rcs* (1B) command.) *co* with locking requires the caller to be on the access list of the RCS file, unless he is the owner of the file or the superuser, or the access list is empty. *co* without locking is not subject to accesslist restrictions.

A revision is selected by number, checkin date/time, author, or state. If none of these options are specified, the latest revision on the trunk is retrieved. When the options are applied in combination, the latest revision that satisfies all of them is retrieved. The options for date/time, author, and state retrieve a revision on the selected branch. The selected branch is either derived from the revision number (if given), or is the highest branch on the trunk. A revision number may be attached to one of the options **-l**, **-p**, **-q**, or **-r**.

A *co* command applied to an RCS file with no revisions creates a zero-length file. *co* always performs keyword substitution (see below).

OPTIONS

-l[*rev*] locks the checked out revision for the caller. If omitted, the checked out revision is not locked. See option **-r** for handling of the revision number rev.

-p[*rev*] prints the retrieved revision on the std. output rather than storing it in the working file. This option is useful when *co* is part of a pipe.

-q[*rev*] quiet mode; diagnostics are not printed.

-d*date* retrieves the latest revision on the selected branch whose checkin date/time is less than or equal to date. The date and time may be given in free format and are converted to local time. Examples of formats for date:

22-April-1982, 17:20-CDT,
2:25 AM, Dec. 29, 1983,
Tue-PDT, 1981,
4pm Jul 21 (free format),
Fri, April 16 15:52:25 EST 1982 (output of ctime).

Most fields in the date and time may be defaulted. *co* determines the defaults in the order year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date "20, 10:30" defaults to 10:30:00 of the 20th of the current month and current year. The date/time must be quoted if it contains spaces.

-r[*rev*] retrieves the latest revision whose number is less than or equal to **rev**. If **rev** indicates a branch rather than a revision, the latest revision on that branch is retrieved. **Rev** is composed of one or more numeric or symbolic fields separated by '.'. The numeric equivalent of a symbolic field is specified with the **-n** option of the commands *ci* and *rcs*.

-s*state* retrieves the latest revision on the selected branch whose state is set to state.

-w[*login*] retrieves the latest revision on the selected branch which was checked in by the user with login name **login**. If the argument **login** is omitted, the caller's login is assumed.

-j*joinlist* generates a new revision which is the join of the revisions on joinlist. Joinlist is a comma-separated list of pairs of the form rev2:rev3, where rev2 and rev3 are (symbolic or numeric) revision numbers. For the initial such pair, rev1 denotes the revision selected by the options **-l**, ..., **-w**. For all other pairs, rev1 denotes the

sion generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, **co** joins revisions rev1 and rev3 with respect to rev2. This means that all changes that transform rev2 into rev1 are applied to a copy of rev3. This is particularly useful if rev1 and rev3 are the ends of two branches that have rev2 as a common ancestor. If rev1 < rev2 < rev3 on the same branch, joining generates a new revision which is like rev3, but with all changes that lead from rev1 to rev2 undone. If changes from rev2 to rev1 overlap with changes from rev2 to rev3, *co* prints a warning and includes the overlapping sections, delimited by the lines <<<<<<< rev1, =======, and >>>>>>> rev3.

For the initial pair, rev2 may be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. If the option **-l** is present, the initial rev1 is locked.

KEYWORD SUBSTITUTION

Strings of the form $keyword$ and $keyword:...$ embedded in the text are replaced with strings of the form $keyword: value $, where keyword and value are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form $keyword$. On checkout, *co* replaces these strings with strings of the form $keyword: value $. If a revision containing strings of the latter form is checked back in, the value fields will be replaced during the next checkout. Thus, the keyword values are automatically updated on checkout.

Keywords and their corresponding values:

$Author$ The login name of the user who checked in the revision.

$Date$ The date and time the revision was checked in.

$Header$ A standard header containing the RCS file name, the revision number, the date, the author, and the state.

$Locker$ The login name of the user who locked the revision (empty if not locked).

Log The log message supplied during checkin, preceded by a header containing the RCS file name, the revision number, the author, and the date. Existing log messages are **NOT** replaced. Instead, the new log message is inserted after $Log:...$. This is useful for accumulating a complete change log in a source file.

$Revision$ The revision number assigned to the revision.

$Source$ The full pathname of the RCS file.

$State$ The state assigned to the revision with *rcs* **-s** or *ci* **-s**.

DIAGNOSTICS

The RCS file name, the working file name, and the revision number retrieved are written to the diagnostic output. The exit status always refers to the last file checked out, and is 0 if the operation was successful, 1 otherwise.

EXAMPLES

Suppose the current directory contains a subdirectory **RCS** with an RCS file **io.c,v**. Then all of the following commands retrieve the latest revision from **RCS/io.c,v** and store it into **io.c**.

```
co io.c; co RCS/io.c,v; co io.c,v;
co io.c RCS/io.c,v; co io.c io.c,v;
co RCS/io.c,v io.c; co io.c,v io.c;
```

FILE MODES

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless the file is checked out unlocked and locking is set to strict (see *rcs* (1B)).

If a file with the name of the working file exists already and has write permission, *co* aborts the checkout if **-q** is given, or asks whether to abort if **-q** is not given. If the existing working file is not writable, it is deleted before the checkout.

FILES

The caller of the command must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory which contains the RCS file.

A number of temporary files are created. A semaphore file is created in the directory of the RCS file to prevent simultaneous update.

SEE ALSO

ci(1B), ident(1B), rcs(1B), rcsdiff(1B), rcsintro(1B), rcsmerge(1B), rlog(1B), rcsfile(5B), sccstorcs(8B).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

LIMITATIONS

The option **-d** gets confused in some circumstances, and accepts no date before 1970. There is no way to suppress the expansion of keywords, except by writing them differently. In *nroff* and *troff*, this is done by embedding the null-character '\&' into the keyword.

BUGS

The option **-j** does not work for files that contain lines with a single '.'.

NAME

diff - differential file and directory comparator

SYNOPSIS

diff [-l] [-r] [-s] [-cefhn] [-biwt] dir1 dir2
diff [-cefhn] [-biwt] file1 file2
diff [-D*string*] [-biw] file1 file2

DESCRIPTION

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

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

There are several options for output format; the default output format contains lines of these forms:

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

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

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

OPTIONS

Options when comparing directories are:

-l	long output format; each text file *diff* is piped through *pr* (1) to paginate it, other differences are remembered and summarized after all text file differences are reported.
-r	causes application of *diff* recursively to common subdirectories encountered.
-s	causes *diff* to report files which are the same, which are otherwise not mentioned.
-S*name*	starts a directory *diff* in the middle beginning with file name.

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

-e produces a script of a, c and d commands for the editor *ed*, which will recreate file2 from file1. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

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

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

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

-n produces a script similar to that of **-e**, but in the opposite order and with a count of changed lines on each insert or delete command. This is the form used by *rcsdiff* (1B).

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

Changes which lie within <context> lines of each other are grouped together on output. (This is a change from the previous *diff* **-c** but the resulting output is usually much easier to interpret.)

-h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

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

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

-w is similar to **-b** but causes whitespace (blanks and tabs) to be totally ignored. E.g., "if (a == b)" will compare equal to "if(a==b)".

-i ignores the case of letters. E.g., "A" will compare equal to "a".

-t will expand tabs in output lines. Normal or **-c** output adds character(s) to the front of each line which may screw up the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

FILES

/tmp/d?????	
/usr/lib/diffh	for **-h**
/bin/diff	for directory diffs
/bin/pr	

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff3(1B)

DIAGNOSTICS

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

BUGS

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

When comparing directories with the **-b**, **-w** or **-i** options specified, *diff* first compares the files ala *cmp*, and then decides to run the *diff* algorithm if they are not equal.

This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

diff3 [-exEX3] file1 file2 file3

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

```
==== all three files differ
====1 file1 is different
====2 file2 is different
====3 file3 is different
```

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f : n1 a Text is to be appended after line number n1 in file f, where f = 1, 2, or 3.

f : n1 , n2 c Text is to be changed in the range line n1 to line n2. If n1 = n2, the range may be abbreviated to n1.

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

Under the **-e** option, *diff3* publishes a script for the editor ed that will incorporate into file1 all changes between file2 and file3, i.e. the changes that normally would be flagged ==== and ====3. Option **-x** (-3) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to 'file1'.

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

The **-E** and **-X** are similar to **-e** and **-x**, respectively, but treat overlapping changes (i.e., changes that would be flagged with ==== in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by "<<<<<<" and ">>>>>>" lines.

For example, suppose lines 7-8 are changed in both file1 and file2. Applying the edit script generated by the command

```
"diff3 -E file1 file2 file3"
```

to file1 results in the file:

```
lines 1-6
of file1
<<<<<<< file1
lines 7-8
of file1
=======
lines 7-8
of file3
>>>>>>> file3
rest of file1
```

The **-E** option is used by RCS *merge* (1B) to insure that overlapping changes in the merged files are preserved and brought to someone's attention.

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(1B)

BUGS

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

NAME

ident - identify files

SYNOPSIS

ident file ...

DESCRIPTION

ident searches the named files for all occurrences of the pattern $keyword:...$, where keyword is one of

```
Author
Date
Header
Locker
Log
Revision
Source
State
```

These patterns are normally inserted automatically by the RCS command *co* (1), but can also be inserted manually.

ident works on text files as well as object files. For example, if the C program in file **f.c** contains

```
char rcsid[] = "$Header: Header information $";
```

and **f.c** is compiled into **f.o**, then the command

```
ident f.c f.o
```

will print

```
f.c:
$Header: Header information $
f.o:
$Header: Header information $
```

SEE ALSO

ci(1B), co(1B), rcs(1B), rcsdiff(1B), rcsintro(1B), rcsmerge(1B), rlog(1B), rcsfile(5B).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

NAME

merge - three-way file merge

SYNOPSIS

merge [-p] file1 file2 file3

DESCRIPTION

merge incorporates all changes that lead from file2 to file3 into file1. The result goes to std. output if **-p** is present, into file1 otherwise. *merge* is useful for combining separate changes to an original. Suppose file2 is the original, and both file1 and file3 are modifications of file2. Then *merge* combines both changes.

An overlap occurs if both file1 and file3 have changes in a common segment of lines. *merge* prints how many overlaps occurred, and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, the user should edit the result and delete one of the alternatives.

SEE ALSO

diff3(1B), diff(1B), rcsmerge(1B), co(1B).

MERGE(1B)

NAME

rcs - change RCS file attributes

SYNOPSIS

rcs [options] file ...

DESCRIPTION

rcs creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For *rcs* to work, the caller's login name must be on the access list, except if the access list is empty, the caller is the owner of the file or the superuser, or the **-i** option is present.

Files ending in **,v** are RCS files, all others are working files. If a working file is given, *rcs* tries to find the corresponding RCS file first in directory **./RCS** and then in the current directory, as explained in *co* (1B).

OPTIONS

-i creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, *rcs* tries to place it first into the subdirectory **./RCS**, and then into the current directory. If the RCS file already exists, an error message is printed.

-a*logins* appends the login names appearing in the comma-separated list logins to the access list of the RCS file.

-A*oldfile* appends the access list of oldfile to the access list of the RCS file.

-e[*logins*] erases the login names appearing in the comma-separated list logins from the access list of the RCS file. If logins is omitted, the entire access list is erased.

-c*string* sets the comment leader to string. The comment leader is printed before every log message line generated by the keyword Log during checkout (see *co*). This is useful for programming languages without multi-line comments. During rcs **-i** or initial *ci*, the comment leader is guessed from the suffix of the working file.

-l[*rev*] locks the revision with number rev. If a branch is given, the latest revision on that branch is locked. If rev is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with *ci* or *rcs* **-u** (see below).

-u[*rev*] unlocks the revision with number rev. If a branch is given, the latest revision on that branch is unlocked. If rev is omitted, the latest lock held by the caller is removed. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the

original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single '.' or CTRL-D.

-L sets locking to strict. Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.

-U sets locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should **not** be used for files that are shared. The default (**-L** or **-U**) is determined by your system administrator.

-n*name*[**:***rev*] associates the symbolic name name with the branch or revision rev. *rcs* prints an error message if name is already associated with another number. If rev is omitted, the symbolic name is deleted.

-N*name*[**:***rev*] same as **-n**, except that it overrides a previous assignment of name.

-o*range* deletes ("outdates") the revisions given by range. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form rev1-rev2 means revisions rev1 to rev2 on the same branch, -rev means from the beginning of the branch containing rev up to and including rev, and rev- means from revision rev to the end of the branch containing rev. None of the outdated revisions may have branches or locks.

-q quiet mode; diagnostics are not printed.

-s*state*[**:***rev*] sets the state attribute of the revision rev to state. If rev is omitted, the latest revision on the trunk is assumed; If rev is a branch number, the latest revision on that branch is assumed. Any identifier is acceptable for state. A useful set of states is Exp (for experimental), Stab (for stable), and Rel (for released). By default, *ci* sets the state of a revision to Exp.

-t[***txtfile***] writes descriptive text into the RCS file (deletes the existing text). If txtfile is omitted, *rcs* prompts the user for text supplied from the std. input, terminated with a line containing a single '.' or CTRL-D. Otherwise, the descriptive text is copied from the file txtfile. If the **-i** option is present, descriptive text is requested even if **-t** is not given. The prompt is suppressed if the std. input is not a terminal.

DIAGNOSTICS

The RCS file name and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

FILES

The caller of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. *rcs* creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, *rcs* always creates a new file. On successful completion, *rcs* deletes the old one and renames the new one. This strategy makes links to RCS files useless.

SEE ALSO

co(1B), ci(1B), ident(1B), rcsdiff(1B), rcsintro(1B), rcsmerge(1B), rlog(1B), rcsfile(5B), sccstorcs(8B).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

NAME

rcsdiff - compare RCS revisions

SYNOPSIS

rcsdiff [-biwt] [-cefhn] [-r*rev1*] [-r*rev2*] file ...

DESCRIPTION

rcsdiff runs *diff* (1B) to compare two revisions of each RCS file given. A file name ending in **,v** is an RCS file name, otherwise a working file name. *rcsdiff* derives the working file name from the RCS file name and vice versa, as explained in *co* (1B). Pairs consisting of both an RCS and a working file name may also be specified.

All options except **-r** have the same effect as described in *diff* (1B).

If both *rev1* and *rev2* are omitted, *rcsdiff* compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what you changed since the last checkin.

If rev1 is given, but *rev2* is omitted, *rcsdiff* compares revision *rev1* of the RCS file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, *rcsdiff* compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* may be given numerically or symbolically.

EXAMPLES

The command

```
rcsdiff f.c
```

runs *diff* on the latest trunk revision of RCS file **f.c,v** and the contents of working file **f.c.**

SEE ALSO

ci(1B), co(1B), diff(1B), ident(1B), rcs(1B), rcsintro(1B), rcsmerge(1B), rlog(1B), rcsfile(5B).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

NAME

rcsintro - introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of text files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, form letters, etc.

The basic user interface is extremely simple. The novice only needs to learn two commands: *ci* and *co*. *ci*, short for "checkin", deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. *co*, short for "checkout", retrieves revisions from an RCS file.

SEE ALSO

ci(1B), co(1B), ident(1B), merge(1B), rcs(1B), rcsdiff(1B), rcsmerge(1B), rlog(1B), rcsfile(5B).

RCSINTRO(1B)

NAME

rcsmerge - merge RCS revisions

SYNOPSIS

rcsmerge -r*rev1* [-r*rev2*] [-p] file

DESCRIPTION

rcsmerge incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If **-p** is given, the result is printed on the std. output, otherwise the result overwrites the working file.

A file name ending in **,v** is an RCS file name, otherwise a working file name. *merge* derives the working file name from the RCS file name and vice versa, as explained in *co* (1B). A pair consisting of both an RCS and a working file name may also be specified.

rev1 may not be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* may be given numerically or symbolically.

rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained in *co* **-j**. The command is useful for incorporating changes into a checked-out revision.

EXAMPLES

Suppose you have released revision 2.8 of **f.c**. Assume furthermore that you just completed revision 3.4, when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file **f.c** and execute

```
rcsmerge -p -r2.8 -r3.4 f.c >f.merged.c
```

Then examine **f.merged.c**. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute *co* **-j**:

```
ci -r2.8.1.1 f.c
co -r3.4 -j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in **f.c**.

```
rcsmerge -r2.8 -r2.4 f.c
```

Note the order of the arguments, and that **f.c** will be overwritten.

SEE ALSO

ci(1B), co(1B), merge(1B), ident(1B), rcs(1B), rcsdiff(1B), rlog(1B), rcsfile(5).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

BUGS

rcsmerge does not work for files that contain lines with a single '.'.

-r*revisions* prints information about revisions given in the comma-separated list revisions of revisions and ranges. A range rev1-rev2 means revisions rev1 to rev2 on the same branch, -rev means revisions from the beginning of the branch up to and including rev, and rev- means revisions starting with rev to the end of the branch containing rev. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

-s*states* prints information about revisions whose state attributes match one of the states given in the comma-separated list states.

-w[*logins*] prints information about revisions checked in by users with login names appearing in the comma-separated list logins. If logins is omitted, the user's login is assumed.

rlog prints the intersection of the revisions selected with the options **-d**, **-l**, **-s**, **-w**, intersected with the union of the revisions selected by **-b** and **-r**.

EXAMPLES

```
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v
```

The first command prints the names of all RCS files in the subdirectory RCS which have locks. The second command prints the headers of those files, and the third prints the headers plus the log messages of the locked revisions. The last command prints complete information.

DIAGNOSTICS

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

SEE ALSO

ci(1B), co(1B), ident(1B), rcs(1B), rcsdiff(1B), rcsintro(1B), rcsmerge(1B), rcsfile(5B), sccstorcs(8B).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

3

NAME

cu - call another UNIX system

SYNOPSIS

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

DESCRIPTION

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

cu accepts the following options and arguments:

-s*speed* Specifies the transmission speed (300, 1200, 2400, 4800, 9600); The default value is "Any" speed which will depend on the order of the lines in the **/usr/lib/uucp/Devices** file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

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

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

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

-d Causes diagnostic traces to be printed.

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

-n	For added security, will prompt the user to provide the telephone number to be dialed rather than taking it from the command line.
-e	Designates that even parity is to be generated for data sent to the remote system.
telno	When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.
systemname	A uucp system name may be used rather than a telephone number; in this case, *cu* will obtain an appropriate direct line or telephone number from **/usr/lib/uucp/Systems.** Note: the *systemname* option should not be used in conjunction with the **-l** and **-s** options as *cu* will connect to the first available line for the system name specified, ignoring the requested line and speed.

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

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

~.	terminate the conversation.
~!	escape to an interactive shell on the local system.
~! *cmd...*	run *cmd* on the local system (via *sh* **-c**).
~$ *cmd...*	run *cmd* locally and send its output to the remote system.
~%cd	change the directory on the local system. Note: **~!cd** will cause the command to be run by a sub-shell, probably not what was intended.
~%take *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%put *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places. For both **~%take** and **put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
~~ *line*	send the line ~ *line* to the remote system.

~%break	transmit a **BREAK** to the remote system (which can also be specified as **~%b**).
~%debug	toggles the **-d** debugging option on or off (which can also be specified as **~%d**).
~t	prints the values of the termio structure variables for the user's terminal (useful for debugging).
~l	prints the values of the termio structure variables for the remote communication line (useful for debugging).
~%nostop	toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with **~.**

Data from the remote is diverted (or appended, if **>>** is used) to *file* on the local system. The trailing **~>** marks the end of the diversion.

The use of **~%put** requires *stty* (1) and *cat* (1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Back-slashes are inserted at appropriate places.

The use of **~%take** requires the existence of *echo* (1) and *cat* (1) on the remote system. Also, *tabs* mode (See *stty* (1)) should be set on the remote system if tabs are to be copied without expansion to spaces.

When *cu* is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using **~~.** Executing a tilde command reminds the user of the local system uname. For example, uname can be executed on Z, X, and Y as follows:

```
uname
z
~[X]!uname
X
~~[Y]!uname
Y
```

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

EXAMPLES

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

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

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

To login to a system connected by a direct line:

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

or

```
cu -l ttyXX
```

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

```
cu -s1200 -l ttyXX
```

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

```
cu -l culXX 9=12015551212
```

To use a system name:

```
cu systemname
```

FILES

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

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uucp(1C), uname(1).

DIAGNOSTICS

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

WARNINGS

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

BUGS

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

NAME

uucp, uulog, uuname - UNIX-to-UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file
uulog [options] **-s system**
uulog [options] system
uulog [options] **-f system**
uuname [**-l**] [**-c**]

DESCRIPTION

uucp

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

```
system-name!path-name
```

where *system-name* is taken from a list of system names that *uucp* knows about. The *system-name* may also be a list of names such as

```
system-name!system-name!...!system-name!path-name
```

in which case an attempt is made to send the file via the specified route, to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to foward information (see WARNINGS below for restrictions).

The shell metacharacters **?**, * and **[...]** appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

1. a full path name;
2. a path name preceded by ***~ user*** where *user* is a login name on the specified system and is replaced by that user's login directory;
3. a path name preceded by ***~/ destination*** where *destination* is appended to **/usr/spool/uucppublic ;** (NOTE: This destination will be treated as a file name unless more than one file is being transfered by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ~/dan/ as the destination will make the directory /usr/spool/uucppublic/dan if it does not exist and put the requested file(s) in that directory).
4. anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

If a simple *~user* destination is inaccessible to *uucp*, data is copied to a spool directory and the user is notified by *mail* (1).

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod* (2)).

The following options are interpreted by *uucp :*

-c	Do not copy local file to the spool directory for transfer to the remote machine (default).
-C	Force the copy of local files to the spool directory for transfer.
-d	Make all necessary directories for the file copy (default).
-f	Do not make intermediate directories for the file copy.
-g*grade*	*grade* is a single letter/number; lower ascii sequence characters will cause the job to be transmitted earlier during a particular conversation.
-j	Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.
-m	Send mail to the requester when the copy is completed.
-n*user*	Notify *user* on the remote system that a file was sent.
-r	Do not start the file transfer, just queue the job.
-s*file*	Report status of the transfer to *file*. Note that the *file* must be a full path name.
-x*debug_level*	Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiles with -DSMALL.)

uulog

uulog queries a log file of *uucp* or *uuxqt* transactions in a file **/usr/spool/uucp/.Log/uucico/*system*,** or **/usr/spool/uucp/.Log/uuxqt/*system*.**

The options cause *uulog* to print logging information:

-s*sys*	Print information about file transfer work involving system *sys*.
-f*system*	Does a *tail* **-f** of the file transfer log for *system*. (You must hit BREAK to exit this function.) Other options used in conjunction with the above:
-x	Look in the *uuxqt* log file for the given system.
- *number*	Indicates that a *tail* command of *number* lines should be executed.

uuname

uuname lists the names of systems known to *uucp*. The **-c** option returns the names of systems known to *cu*. (The two lists are the same, unless your

machine is using different *Systems* files for *cu* and *uucp*. See the *Sysfiles* file.) The **-l** option returns the local system name.

FILES

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending (**/usr/spool/uucppublic**)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uustat(1C), uux(1C), uuxqt(1M).

chmod(2) in the *Programmer's Reference Manual.*

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to **~/).**

All files received by *uucp* will be owned by *uucp.*

The **-m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **? * [...]** will not activate the **-m** option.

The forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent by *uucp*. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uustat - uucp status inquiry and job control

SYNOPSIS

uustat [-a]
uustat [-m]
uustat [-p]
uustat [-q]
uustat [-k*jobid*]
uustat [-r*jobid*]
uustat [-s*system*] [-u*user*]

DESCRIPTION

uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

-a Output all jobs in queue.

-m Report the status of accessibility of all machines.

-p Execute a *ps* **-flp** for all the process-ids that are in the lock files.

-q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of **C** or **X** files, it is the age in days of the oldest **C./X.** file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the **-q** option:

```
eagle      3C   04/07-11:07     NO DEVICES AVAILABLE
mh3bs3     2C   07/07-10:42     SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

-k*jobid* Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

-r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from

deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with *uustat :*

-s*sys* Report the status of all *uucp* requests for remote system *sys*.

-u*user* Report the status of all *uucp* requests issued by *user.*

Output for both the **-s** and **-u** options has the following format:

```
eaglen0000   4/07-11:01:03   (POLL)
eagleN1bd7   4/07-11:07      Seagledan  522 /usr/dan/A
eagleC1bd8   4/07-11:07      Seagledan  59 D.3b2a12ce4924
                 4/07-11:07 Seagledan  rmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).

When no options are given, uustat outputs the status of all *uucp* requests issued by the current user.

FILES

/usr/spool/uucp/* spool directories

SEE ALSO

uucp(1C).

NAME

uuto, uupick - public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [options] source-files destination
uupick [**-s** system]

DESCRIPTION

uuto sends *source-files* to *destination*. *uuto* uses the *uucp* (1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

```
system!user
```

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *User* is the login name of someone on the specified system.

Two *options* are available:

-p	Copy the source file into the spool directory before transmission.
-m	Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. By default this directory is **/usr/spool/uucppublic**. Specifically the files are sent to

```
PUBDIR/receive/user/mysystem/files.
```

The destined recipient is notified by *mail* (1) of the arrival of files.

uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

```
from system: [file file-name] [dir dirname] ?
```

uupick then reads a line from the standard input to determine the disposition of the file:

new-line	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete path name (in which **$HOME** is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT **(control-d)**	Same as **q**.

! *command*	Escape to the shell to do *command*.
*	Print a command summary.

uupick invoked with the **-s*system*** option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR	/usr/spool/uucppublic	public directory

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C).

uucleanup(1M) in the *Administrator's Reference Manual*.

WARNINGS

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

NAME

uux - UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

NOTE: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permiting only the receipt of mail (see *mail* (1)). (Remote execution permissions are defined in **/usr/lib/uucp/Permissions.)**

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-namel.* A null *system-name* is interpreted as the local system.

File names may be one of

1. a full path name;
2. a path name preceded by ~ ***xxx*** where *xxx* is a login name on the specified system and is replaced by that user's login directory;
3. anything else is prefixed by the current directory.

As an example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

will get the *file1* and *file2* files from the "usg" and "pwba" machines, execute a *diff* (1) command and put the results in *file.diff* in the local **PUBDIR/dan/** directory.

Any special shell characters such as **<>;|** should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!cut -f1 b!/usr/file \(c!/usr/file\)
```

gets /usr/file from system "b" and sends it to system "a", performs a *cut* command on that file and sends the result of the *cut* command to system "c".

uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **-n** option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux :*

-	The standard input to *uux* is made the standard input to the *command-string*.
-aname	Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
-b	Return whatever standard input was provided to the *uux* command if the exit status is non-zero.
-c	Do not copy local file to the spool directory for transfer to the remote machine (default).
-C	Force the copy of local files to the spool directory for transfer.
-ggrade	*grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.
-j	Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.
-n	Do not notify the user if the command fails.
-p	Same as -: The standard input to *uux* is made the standard input to the *command-string*.
-r	Do not start the file transfer, just queue the job.
-sfile	Report status of the transfer in *file*.
-xdebug_level	Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.
-z	Send success notification to the user.

FILES

/usr/lib/uucp/spool	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

WARNINGS

Only the first command of a shell pipeline may have a *system-name!*. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will **NOT** work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work. (If *diff* is a permitted command.)

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using *uux*. However, if the requestor is root, and the directory is not searchable by "other", the request will fail.

UUX(1C)

4

NAME

intro - introduction to games

DESCRIPTION

This section describes the recreational and educational programs found in the directory **/usr/games**.

The availability of these programs may vary from system to system.

INTRO(6)

NAME

arithmetic - provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [+-x/] [range]

DESCRIPTION

arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; +-x/ respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +-

range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of range. Default range is 10.

At the start, all numbers less than or equal to range are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts de novo. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

back - the game of backgammon

SYNOPSIS

/usr/games/back

DESCRIPTION

back is a program which provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select. In addition to selecting the opponent's level, you may also indicate that you would like to roll your own dice during your turns (for the superstitious players). You will also be given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated.

The points are numbered 1-24, with 1 being white's extreme inner table, 24 being brown's inner table, 0 being the bar for removed white pieces and 25 the bar for brown. For details on how moves are expressed, type **y** when *back* asks **Instructions?** at the beginning of the game. When *back* first asks **Move?**, type **?** to see a list of move options other than entering your numerical move.

When the game is finished, *back* will ask you if you want the log. If you respond with **y**, *back* will attempt to append to or create a file **back.log** in the current directory.

FILES

/usr/games/lib/backrules	rules file
/tmp/b*	log temp file
back.log	log file

BUGS

The only level really worth playing is "expert", and it only plays the forward game.

back will complain loudly if you attempt to make too many moves in a turn, but will become very silent if you make too few.

Doubling is not implemented.

NAME

bj - the game of black jack

SYNOPSIS

/usr/games/bj

DESCRIPTION

bj is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

The bet is $2 every hand.

A player "natural" (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a "push" (no money exchange).

If the dealer has an ace up, the player is allowed to make an "insurance" bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.

If the player is dealt two cards of the same value, he is allowed to "double". He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)

If a dealt hand has a total of ten or eleven, the player may "double down". He may double the bet ($2 to $4) and receive exactly one more card on that hand.

Under normal play, the player may "hit" (draw a card) as long as his total is not over twenty-one. If the player "busts" (goes over twenty-one), the dealer wins the bet.

When the player "stands" (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.

If both player and dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions will be shown at appropriate times. Each question is answered by **y** followed by a new-line for "yes", or just new-line for "no".

? (means, "do you want a hit?")

Insurance?

Double down?

Every time the deck is shuffled, the dealer so states and the "action" (total bet) and "standing" (total won or lost) is printed. To exit, hit the interrupt key (DEL) and the action and standing will be printed.

NAME

craps - the game of craps

SYNOPSIS

/usr/games/craps

DESCRIPTION

craps is a form of the game of craps that is played in Las Vegas. The program simulates the roller, while the user (the player) places bets. The player may choose, at any time, to bet with the roller or with the House. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a "bankroll" of $2,000.

The program prompts with:

```
bet?
```

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with **bet?** until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The first roll is the roll immediately following a bet:

1. On the first roll:

   7 or 11 wins for the roller;

   2, 3, or 12 wins for the House;

   any other number
   is the point, roll again (Rule 2 applies).

2. On subsequent rolls:

   point roller wins;

   7 House wins;

   any other number
   roll again.

If a player loses the entire bankroll, the House will offer to lend the player an additional $2,000. The program will prompt:

```
marker?
```

A yes (or **y**) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

```
Repay marker?
```

A reply of yes (or **y**) indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

```
How many?
```

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program will prompt with **How many?** until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the total amount of money borrowed will be automatically repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

```
New game?
```

to give the House a chance to win back its money.

Any reply other than yes is considered to be a no (except in the case of **bet?** or **How many?**). To exit, send an interrupt (break), DEL, or CTRL-D. The program will indicate whether the player won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

NAME

fish - play "Go Fish"

SYNOPSIS

/usr/games/fish

DESCRIPTION

fish plays the game of Go Fish, a childrens' card game. The Object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; The default is pretty dumb.

FISH(6)

NAME

fortune - print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [-] [**-wslao**] ..[file]

DESCRIPTION

fortune with no arguments prints out a random adage. The flags mean:

-w	Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.
-s	Short messages only.
-l	Long messages only.
-o	Choose from an alternate list of adages, often used for potentially offensive ones.
-a	Choose from either list of adages.

The user may specify a file of adages. This file must be created by *strfile* (6), and be given by the user as it file. Only one such file may be named, subsequent ones are ignored.

FILES

/usr/games/lib/fortunes.dat

AUTHOR

Ken Arnold

SEE ALSO

strfile(6)

NAME

hangman - guess the word

SYNOPSIS

/usr/games/hangman [arg]

DESCRIPTION

hangman chooses a word at least seven letters long from a dictionary. The user is to guess letters one at a time.

The optional argument arg names an alternate dictionary.

FILES

/usr/lib/w2006

BUGS

Hyphenated compounds are run together.

HANGMAN(6)

NAME

jotto - secret word game

SYNOPSIS

/usr/games/jotto [-p]

DESCRIPTION

jotto is a word guessing game. You try to guess the computer's secret word before it guesses yours. Clues are obtained by entering probe words. For example, if the computer's secret word is "brown" and you probe with "stare", it will reply "1" indicating that there is one letter in common between your probe and the secret word. Double letters count only once unless they appear in both words. For example, if the hidden word is "igloo" and you probe with "broke", the computer will reply "1". But if you probe with "gloom", the computer will respond "4". All secret words and probe words should be non-proper English five-letter words. If the computer guesses your word exactly, please respond with **y**. It will then tell you what its secret word was. The **-p** flag instructs the computer to report its progress in guessing your word.

BUGS

The dictionary contains some unusual words and lacks some common ones.

NAME

maze - generate a maze

SYNOPSIS

/usr/games/maze

DESCRIPTION

Maze prints a maze.

BUGS

Some mazes (especially small ones) have no solutions.

NAME

moo - guessing game

SYNOPSIS

/usr/games/moo

DESCRIPTION

moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A "cow" is a correct digit in an incorrect position. A "bull" is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).

MOO(6)

NAME

quiz - test your knowledge

SYNOPSIS

/usr/games/quiz [-i file] [-t] [category1 category2]

DESCRIPTION

quiz gives associative knowledge tests on various subjects. It asks items chosen from category1 and expects answers from category2, or vice versa. If no categories are specified, *quiz* gives instructions and lists the available categories.

quiz tells a correct answer whenever you type a bare new- line. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **-t** flag specifies "tutorial" mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **-i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      =   category new-line | category : line
category  =   alternate | category | alternate
alternate =   empty | alternate primary
primary   =   character | [ category ] | option
option    =   { category }
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash \ is used as with *sh* (1) to quote syntactically significant characters or to insert transparent new-lines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

FILES

/usr/games/lib/quiz/index
/usr/games/lib/quiz/*

BUGS

The construct "a | ab" does not work in an information file. Use "a{b}".

NAME

ttt, cubic - tic-tac-toe

SYNOPSIS

/usr/games/ttt

DESCRIPTION

ttt is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

TTT(6)

NAME

wump - the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

wump plays the game of "Hunt the Wumpus." A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in People's Computer Company, 2, 2 (November 1973).

BUGS

It will never replace Adventure.

5

Manual references for D-NIX 5.3

This is a reference section for the D-NIX 5.3 operating system. All commands and functions are listed with a reference to the appropriate manual.

A

.out(4)	AT&T Administrator's Reference Manual
a64l(3C)	AT&T Programmer's Reference Manual
abort(3C)	AT&T Programmer's Reference Manual
abs(3C)	AT&T Programmer's Reference Manual
accept(1M)	D-NIX 5.3 Reference Manual
access(2)	AT&T Programmer's Reference Manual
acct(1M)	AT&T Administrator's Reference Manual
acct(2)	AT&T Programmer's Reference Manual
acct(4)	AT&T Administrator's Reference Manual
acctcms(1M)	AT&T Administrator's Reference Manual
acctcom(1)	AT&T User's Reference Manual
acctcon(1M)	AT&T Administrator's Reference Manual
acctcon1(1M)	See acctcon(1M)
acctcon2(1M)	See acctcon(1M)
acctdisk(1M)	See acct(1M)
acctdusg(1M)	See acct(1M)
acctmerg(1M)	AT&T Administrator's Reference Manual
accton(1M)	See acct(1M)
acctprc(1M)	AT&T Administrator's Reference Manual
acctprc1(1M)	See acctprc(1M)
acctprc2(1M)	See acctprc(1M)
acctsh(1M)	AT&T Administrator's Reference Manual
acctwtmp(1M)	See acct(1M)
acos(3M)	See trig(3M)
adb(1)	AT&T Programmer's Reference Manual
admin(1)	AT&T Programmer's Reference Manual
alarm(2)	AT&T Programmer's Reference Manual
ar(1)	AT&T Programmer's Reference Manual
ar(4)	AT&T Administrator's Reference Manual
arithmetic(6)	AT&T User's Reference Manual
as(1)	AT&T Programmer's Reference Manual
ascii(5)	AT&T Administrator's Reference Manual
asctime(3C)	See ctime(3C)
asin(3M)	See trig(3M)
assert(3X)	AT&T Programmer's Reference Manual
at(1)	AT&T User's Reference Manual
atan(3M)	See trig(3M)
atan2(3M)	See trig(3M)
atof(3C)	See strtod(3C)
atoi(3C)	See strtol(3C)
atol(3C)	See strtol(3C)
awk(1)	AT&T User's Reference Manual

B

back(6)	AT&T User's Reference Manual
badblk(1M)	D-NIX 5.3 Reference Manual
banner(1)	D-NIX 5.3 Reference Manual
basename(1)	D-NIX 5.3 Reference Manual
batch(1)	See at(1)
bc(1)	AT&T User's Reference Manual
bdiff(1)	AT&T User's Reference Manual
bessel(3M)	AT&T Programmer's Reference Manual
bfs(1)	AT&T User's Reference Manual
bj(6)	AT&T User's Reference Manual
bootpar(1M)	D-NIX 5.3 Reference Manual
brk(2)	AT&T Programmer's Reference Manual
bsearch(3C)	AT&T Programmer's Reference Manual
bup(1)	D-NIX 5.3 Reference Manual

C

cal(1)	AT&T User's Reference Manual
calendar(1)	AT&T User's Reference Manual
calloc(3C)	See malloc(3C)
calloc(3X)	See malloc(3X)
cancel(1)	D-NIX 5.3 Reference Manual
captoinfo(1M)	AT&T Administrator's Reference Manual
cat(1)	D-NIX 5.3 Reference Manual
cb(1)	AT&T Programmer's Reference Manual
cc(1)	AT&T Programmer's Reference Manual
cd(1)	D-NIX 5.3 Reference Manual
cdc(1)	AT&T Programmer's Reference Manual
ceil(3M)	See floor(3M)
cflow(1)	AT&T Programmer's Reference Manual
chargefee(1M)	See acctsh(1M)
chdir(2)	AT&T Programmer's Reference Manual
checkeq(1)	See eqn(1)
checklist(4)	AT&T Administrator's Reference Manual
chgrp(1)	D-NIX 5.3 Reference Manual
chmod(1)	D-NIX 5.3 Reference Manual
chmod(2)	AT&T Programmer's Reference Manual
chown(1)	D-NIX 5.3 Reference Manual
chown(2)	AT&T Programmer's Reference Manual
chroot(1M)	AT&T Administrator's Reference Manual
chroot(2)	AT&T Programmer's Reference Manual
chrtbl(1)	D-NIX Reference Manual
ci(1B)	AT&T User's Reference Manual
ckpacct(1M)	See acctsh(1M)
clearerr(3S)	See ferror(3S)
clock(3C)	AT&T Programmer's Reference Manual
clone(7)	AT&T Administrator's Reference Manual
close(2)	AT&T Programmer's Reference Manual
closedir(3X)	See directory(3X)

cmp(1)	D-NIX 5.3 Reference Manual
co(1B)	AT&T User's Reference Manual
col(1)	AT&T User's Reference Manual
comb(1)	AT&T Programmer's Reference Manual
comm(1)	AT&T User's Reference Manual
compress(1)	AT&T User's Reference Manual
console(7)	AT&T Administrator's Reference Manual
conv(3C)	AT&T Programmer's Reference Manual
copy(1)	D-NIX 5.3 Reference Manual
core(4)	AT&T Administrator's Reference Manual
cos(3M)	See trig(3M)
cosh(3M)	See sinh(3M)
cp(1)	D-NIX 5.3 Reference Manual
cpio(1)	D-NIX 5.3 Reference Manual
cpio(4)	AT&T Administrator's Reference Manual
cpp(1)	AT&T Programmer's Reference Manual
cpset(1M)	AT&T Administrator's Reference Manual
craps(6)	AT&T User's Reference Manual
creat(2)	AT&T Programmer's Reference Manual
cron(1M)	D-NIX 5.3 Reference Manual
crontab(1)	D-NIX 5.3 Reference Manual
crypt(1)	AT&T User's Reference Manual
crypt(3C)	AT&T Programmer's Reference Manual
crypt(3X)	AT&T Programmer's Reference Manual
csh(1)	AT&T User's Reference Manual
csplit(1)	AT&T User's Reference Manual
ctags(1)	AT&T User's Reference Manual
ctermid(3S)	AT&T Programmer's Reference Manual
ctime(3C)	AT&T Programmer's Reference Manual
ctoa(1)	AT&T Programmer's Reference Manual
ctrace(1)	AT&T Programmer's Reference Manual
ctype(3C)	AT&T Programmer's Reference Manual
cu(1C)	D-NIX 5.3 Reference Manual
curses(3X)	AT&T Programmer's Reference Manual
cuserid(3S)	AT&T Programmer's Reference Manual
cut(1)	D-NIX 5.3 Reference Manual
cxref(1)	AT&T Programmer's Reference Manual

D

date(1)	D-NIX 5.3 Reference Manual
DBon(1)	D-NIX 5.3 Reference Manual
DBscan(1)	D-NIX 5.3 Reference Manual
dc(1)	AT&T User's Reference Manual
dd(1M)	D-NIX 5.3 Reference Manual
delta(1)	AT&T Programmer's Reference Manual

deroff(1)	AT&T User's Reference Manual
devnm(1M)	D-NIX 5.3 Reference Manual
df(1M)	D-NIX 5.3 Reference Manual
diab1130	See machid(1)
diab1320	See machid(1)
diab1420	See machid(1)
diab2340	See machid(1)
diab2420	See machid(1)
diab2430	See machid(1)
diab2440	See machid(1)
diab2450	See machid(1)
dial(3C)	AT&T Programmer's Reference Manual
diff(1)	AT&T User's Reference Manual
diff(1B)	AT&T User's Reference Manual
diff3(1)	AT&T User's Reference Manual
diff3(1B)	AT&T User's Reference Manual
diffmk(1)	AT&T User's Reference Manual
dir(4)	AT&T Administrator's Reference Manual
dircmp(1)	AT&T User's Reference Manual
directory(3X)	AT&T Programmer's Reference Manual
dirent(4)	AT&T Administrator's Reference Manual
dirname(1)	D-NIX 5.3 Reference Manual
disable(1)	D-NIX 5.3 Reference Manual
diskusg(1M)	AT&T Administrator's Reference Manual
dnixcore(4)	D-NIX 5.3 Reference Manual
dmacs(1)	D-NIX 5.3 Reference Manual
dodisk(1M)	See acctsh(1M)
drand48(3C)	AT&T Programmer's Reference Manual
ds90-10	See machid(1)
ds90-11	See machid(1)
ds90-20	See machid(1)
ds90-21	See machid(1)
ds90-30)	See machid(1)
ds90-30s	See machid(1)
ds90-31	See machid(1)
ds90-41	See machid(1)
ds90-45)	See machid(1)
du(1M)	D-NIX 5.3 Reference Manual
dump(1)	AT&T Programmer's Reference Manual
dup(2)	AT&T Programmer's Reference Manual
dup2(3C)	AT&T Programmer's Reference Manual

E

echo(1)	D-NIX 5.3 Reference Manual
ecvt(3C)	AT&T Programmer's Reference Manual
ed(1)	D-NIX 5.3 Reference Manual
edata(3C)	See end(3C)

edit(1)	AT&T User's Reference Manual
egrep(1)	AT&T User's Reference Manual
enable(1)	D-NIX 5.3 Reference Manual
encrypt(3C)	See crypt(3C)
end(3C)	AT&T Programmer's Reference Manual
endgrent(3C)	See getgrent(3C)
endpwent(3C)	See getpwent(3C)
endutent(3C)	See getut(3C)
env(1)	D-NIX 5.3 Reference Manual
environ(5)	AT&T Administrator's Reference Manual
eqn(1)	AT&T User's Reference Manual
erand48(3C)	See drand48(3C)
erf(3M)	AT&T Programmer's Reference Manual
erfc(3M)	See erf(3M)
errdemon(1M)	D-NIX 5.3 Reference Manual
errno(3C)	See perror(3C)
etext(3C)	See end(3C)
eval(1)	D-NIX 5.3 Reference Manual
ex(1)	AT&T User's Reference Manual
exec(1)	D-NIX 5.3 Reference Manual
exec(2)	AT&T Programmer's Reference Manual
exit(1)	D-NIX 5.3 Reference Manual
exit(2)	AT&T Programmer's Reference Manual
exp(3M)	AT&T Programmer's Reference Manual
export(1)	D-NIX 5.3 Reference Manual
expr(1)	D-NIX 5.3 Reference Manual

F

fabs(3M)	See floor(3M)
factor(1)	AT&T User's Reference Manual
false(1)	D-NIX 5.3 Reference Manual
fclose(3S)	AT&T Programmer's Reference Manual
fcntl(2)	AT&T Programmer's Reference Manual
fcntl(5)	AT&T Administrator's Reference Manual
fcvt(3C)	See ecvt(3C)
fdopen(3S)	See fopen(3S)
ferror(3S)	AT&T Programmer's Reference Manual
feof(3S)	See ferror(3S)
fflush(3S)	See fclose(3S)
fgetc(3S)	See getc(3S)
fgetgrent(3C)	See getgrent(3C)
fgetpwent(3C)	See getpwent(3C)
fgets(3S)	See gets(3S)
fgrep(1)	D-NIX 5.3 Reference Manual
file(1)	AT&T User's Reference Manual
filehdr(4)	AT&T Administrator's Reference Manual

fileno(3S)	See ferror(3S)
find(1)	D-NIX 5.3 Reference Manual
fish(6)	AT&T User's Reference Manual
floor(3M)	AT&T Programmer's Reference Manual
fmod(3M)	See floor(3M)
fopen(3S)	AT&T Programmer's Reference Manual
fork(2)	AT&T Programmer's Reference Manual
format(1M)	D-NIX 5.3 Reference Manual
fortune(6)	AT&T User's Reference Manual
fprintf(3S)	See printf(3S)
fputc(3S)	See putc(3S)
fputs(3S)	See puts(3S)
fread(3S)	AT&T Programmer's Reference Manual
free(3C)	See malloc(3C)
free(3X)	See malloc(3X)
freopen(3S)	See fopen(3S)
frexp(3C)	AT&T Programmer's Reference Manual
fscanf(3S)	See scanf(3S)
fsck(1)	D-NIX 5.3 Reference Manual
fscl(1)	D-NIX 5.3 Reference Manual
fseek(3S)	AT&T Programmer's Reference Manual
fsize(1)	D-NIX 5.3 Reference Manual
fspec(4)	AT&T Administrator's Reference Manual
fstab(4)	AT&T Administrator's Reference Manual
fstat(2)	See stat(2)
fstatfs(2)	See statfs(2)
ftell(3S)	See fseek(3S)
ftok(3C)	See stdipc(3C)
ftw(3C)	AT&T Programmer's Reference Manual
fuser(1M)	AT&T Administrator's Reference Manual
fwrite(3S)	See fread(3S)
fwtmp(1M)	AT&T Administrator's Reference Manual

G

gamma(3M)	AT&T Programmer's Reference Manual
gcvt(3C)	See ecvt(3C)
get(1)	AT&T Programmer's Reference Manual
getc(3S)	AT&T Programmer's Reference Manual
getchar(3S)	See getc(3S)
getcwd(3C)	AT&T Programmer's Reference Manual
getdents(2)	AT&T Programmer's Reference Manual
getenv(3C)	AT&T Programmer's Reference Manual
geteuid(2)	See getuid(2)
getegid(2)	See getuid(2)
getgid(2)	See getuid(2)
getgrent(3C)	AT&T Programmer's Reference Manual
getgrgid(3C)	See getgrent(3C)

getgrnam(3C)	See getgrent(3C)
getlogin(3C)	AT&T Programmer's Reference Manual
getmsg(2)	AT&T Programmer's Reference Manual
getopts(1)	D-NIX 5.3 Reference Manual
getopt(3C)	AT&T Programmer's Reference Manual
getpass(3C)	AT&T Programmer's Reference Manual
getpid(2)	AT&T Programmer's Reference Manual
getpgrp(2)	See getpid(2)
getppid(2)	See getpid(2)
getpw(3C)	AT&T Programmer's Reference Manual
getpwent(3C)	AT&T Programmer's Reference Manual
getpwnam(3C)	See getpwent(3C)
getpwuid(3C)	See getpwent(3C)
gets(3S)	AT&T Programmer's Reference Manual
getspent(3X)	AT&T Programmer's Reference Manual
getty(1M)	D-NIX 5.3 Reference Manual
gettydefs(4)	AT&T Administrator's Reference Manual
getuid(2)	AT&T Programmer's Reference Manual
getut(3C)	AT&T Programmer's Reference Manual
getutent(3C)	See getut(3C)
getutid(3C)	See getut(3C)
getutline(3C)	See getut(3C)
getw(3S)	See getc(3S)
glossary(1)	AT&T User's Reference Manual
gmtime(3C)	See ctime(3C)
grep(1)	D-NIX 5.3 Reference Manual
group(4)	AT&T Administrator's Reference Manual
grpck(1M)	See pwck(1M)
gsignal(3C)	See ssignal(3C)

H

hangman(6)	AT&T User's Reference Manual
hashcheck(1)	See spell(1)
hashmake(1)	See spell(1)
hcreate(3C)	See hsearch(3C)
hd(1)	AT&T User's Reference Manual
hdestroy(3C)	See hsearch(3C)
help(1)	AT&T User's Reference Manual
helpadm(1M)	AT&T Administrator's Reference Manual
holidays(4)	AT&T Administrator's Reference Manual
hsearch(3C)	AT&T Programmer's Reference Manual
hypot(3M)	AT&T Programmer's Reference Manual

I

id(1M)	D-NIX 5.3 Reference Manual
ident(1B)	AT&T User's Reference Manual
infocmp(1M)	AT&T Administrator's Reference Manual
init(1M)	D-NIX 5.3 Reference Manual
inittab(4)	AT&T Administrator's Reference Manual
inode(4)	AT&T Administrator's Reference Manual
install(1M)	AT&T Administrator's Reference Manual
ioctl(2)	AT&T Programmer's Reference Manual
ipcrm(1)	D-NIX 5.3 Reference Manual
ipcs(1)	D-NIX 5.3 Reference Manual
isalpha(3C)	See ctype(3C)
is.....(3C)	See ctype(3C)
isatty(3C)	See ttyname(3C)
issue(4)	AT&T Administrator's Reference Manual

J

jn(3M)	See bessel(3M)
join(1)	AT&T User's Reference Manual
jotto(6)	AT&T User's Reference Manual
jrand48(3C)	See drand48(3C)

K

kermit(1)	D-NIX 5.3 Reference Manual
key(4)	D-NIX 5.3 Reference Manual
kill(1)	D-NIX 5.3 Reference Manual
kill(2)	AT&T Programmer's Reference Manual
kmem(7)	See mem(7)

L

l(1)	D-NIX 5.3 Reference Manual
l3tol(3C)	AT&T Programmer's Reference Manual
l64a(3C)	See a64l(3C)
labelit(1M)	D-NIX 5.3 Reference Manual
lastlogin(1M)	See acctsh(1M)
lc(1)	D-NIX 5.3 Reference Manual
lcong48(3C)	See drand48(3C)
ld(1)	AT&T Programmer's Reference Manual
ldaclose(3X)	See ldclose(3X)

ldahread(3X)	AT&T Programmer's Reference Manual
ldaopen(3X)	See ldopen(3X)
ldclose(3X)	AT&T Programmer's Reference Manual
ldexp(3C)	See frexp(3C)
ldfcn(4)	AT&T Administrator's Reference Manual
ldfhread(3X)	AT&T Programmer's Reference Manual
ldgetname(3X)	AT&T Programmer's Reference Manual
ldlinit(3X)	See ldlread(3X)
ldlitem(3X)	See ldlread(3X)
ldlread(3X)	AT&T Programmer's Reference Manual
ldlseek(3X)	AT&T Programmer's Reference Manual
ldnlseek(3X)	See ldlseek(3X)
ldnrseek(3X)	See ldrseek(3X)
ldnsseek(3X)	See ldsseek(3X)
ldohseek(3X)	AT&T Programmer's Reference Manual
ldopen(3X)	AT&T Programmer's Reference Manual
ldnshread(3X)	See ldshread(3X)
ldrseek(3X)	AT&T Programmer's Reference Manual
ldshread(3X)	AT&T Programmer's Reference Manual
ldsseek(3X)	AT&T Programmer's Reference Manual
ldtbindex(3X)	AT&T Programmer's Reference Manual
ldtbread(3X)	AT&T Programmer's Reference Manual
ldtbseek(3X)	AT&T Programmer's Reference Manual
lex(1)	AT&T Programmer's Reference Manual
lfind(3C)	See lsearch(3C)
limits(4)	AT&T Administrator's Reference Manual
line(1)	D-NIX 5.3 Reference Manual
linenum(4)	AT&T Administrator's Reference Manual
link(1M)	AT&T Administrator's Reference Manual
link(2)	AT&T Programmer's Reference Manual
lint(1)	AT&T Programmer's Reference Manual
ln(1)	D-NIX 5.3 Reference Manual
load(1)	D-NIX 5.3 Reference Manual
localtime(3C)	See ctime(3C)
locate(1)	AT&T User's Reference Manual
lockf(3C)	AT&T Programmer's Reference Manual
log(3M)	See exp(3M)
log10(3M)	See exp(3M)
login(1)	D-NIX 5.3 Reference Manual
loginlog(4)	AT&T Administrator's Reference Manual
logname(1)	D-NIX 5.3 Reference Manual
logname(3X)	AT&T Programmer's Reference Manual
longjmp(3C)	See setjmp(3C)
lorder(1)	AT&T Programmer's Reference Manual
lp(1)	D-NIX 5.3 Reference Manual
lpadmin(1M)	D-NIX 5.3 Reference Manual
lpd(1)	D-NIX 5.3 Reference Manual
lpfilter(1M)	D-NIX 5.3 Reference Manual
lpforms(1)	D-NIX 5.3 Reference Manual
lpmove(1M)	D-NIX 5.3 Reference Manual
lppg(1)	D-NIX 5.3 Reference Manual
lpr(1)	D-NIX 5.3 Reference Manual

lpsched(1)	D-NIX 5.3 Reference Manual
lpshut(1M)	D-NIX 5.3 Reference Manual
lpstat(1)	D-NIX 5.3 Reference Manual
lpsubmit(1)	D-NIX 5.3 Reference Manual
lpusers(1M)	D-NIX 5.3 Reference Manual
lrand48(3C)	See drand48(3C)
ls(1)	D-NIX 5.3 Reference Manual
lsearch(3C)	AT&T Programmer's Reference Manual
lseek(2)	AT&T Programmer's Reference Manual
ltol3(3C)	See l3tol(3C)

M

m4(1)	AT&T Programmer's Reference Manual
machid(1)	D-NIX 5.3 Reference Manual
mail(1)	D-NIX 5.3 Reference Manual
mailx(1)	AT&T User's Reference Manual
make(1)	AT&T Programmer's Reference Manual
makekey(1)	AT&T User's Reference Manual
mallinfo(3X)	See malloc(3X)
malloc(3C)	AT&T Programmer's Reference Manual
malloc(3X)	AT&T Programmer's Reference Manual
mallopt(3X)	See malloc(3X)
man(1)	AT&T User's Reference Manual
math(5)	AT&T Administrator's Reference Manual
matherr(3M)	AT&T Programmer's Reference Manual
maze(6)	AT&T User's Reference Manual
mc68k	See machid(1)
mc68000	See machid(1)
mc68020	See machid(1)
mc68030	See machid(1)
mc68040	See machid(1)
mem(7)	AT&T Administrator's Reference Manual
memccpy(3C)	See memory(3C)
memchr(3C)	See memory(3C)
memcmp(3C)	See memory(3C)
memcpy(3C)	See memory(3C)
memory(3C)	AT&T Programmer's Reference Manual
memset(3C)	See memory(3C)
merge(1B)	AT&T User's Reference Manual
mesg(1)	D-NIX 5.3 Reference Manual
mgetty(1M)	D-NIX 5.3 Reference Manual
mkcfig(1M)	D-NIX 5.3 Reference Manual
mkcont(1)	D-NIX 5.3 Reference Manual
mkdir(1)	D-NIX 5.3 Reference Manual
mkdir(2)	AT&T Programmer's Reference Manual
mkfs(1M)	D-NIX 5.3 Reference Manual
mknod(1M)	D-NIX 5.3 Reference Manual
mknod(2)	AT&T Programmer's Reference Manual

mknodm(1)	D-NIX 5.3 Reference Manual
mksort(1)	D-NIX 5.3 Reference Manual
mktemp(3C)	AT&T Programmer's Reference Manual
mkuser(1M)	D-NIX 5.3 Reference Manual
mntchk(1M)	D-NIX 5.3 Reference Manual
mnttab(4)	AT&T Administrator's Reference Manual
modf(3C)	See frexp(3C)
monacct(1M)	See acctsh(1M)
monitor(3C)	AT&T Programmer's Reference Manual
moo(6)	AT&T User's Reference Manual
mount(1M)	D-NIX 5.3 Reference Manual
mount(2)	AT&T Programmer's Reference Manual
mrand48(3C)	See drand48(3C)
msgctl(2)	AT&T Programmer's Reference Manual
msgget(2)	AT&T Programmer's Reference Manual
msgop(2)	AT&T Programmer's Reference Manual
mv(1)	D-NIX 5.3 Reference Manual
mvdir(1M)	D-NIX 5.3 Reference Manual

N

nawk(1)	AT&T User's Reference Manual
neqn(1)	See eqn(1)
newform(1)	AT&T User's Reference Manual
newgrp(1)	D-NIX 5.3 Reference Manual
news(1)	AT&T User's Reference Manual
nice(1)	D-NIX 5.3 Reference Manual
nice(2)	AT&T Programmer's Reference Manual
nl(1)	AT&T User's Reference Manual
nlist(3C)	AT&T Programmer's Reference Manual
nm(1)	AT&T Programmer's Reference Manual
nohup(1)	See nice(1)
nrand48(3C)	See drand48(3C)
nroff(1)	AT&T User's Reference Manual
null(7)	AT&T Administrator's Reference Manual
nulladm(1M)	See acctsh(1M)

O

od(1)	D-NIX 5.3 Reference Manual
open(2)	AT&T Programmer's Reference Manual
opendir(3X)	See directory(3X)
oscore(1)	D-NIX 5.3 Reference Manual

P

pack(1)	AT&T User's Reference Manual
passmgmt(1)	D-NIX 5.3 Reference Manual
passwd(1)	D-NIX 5.3 Reference Manual
passwd(4)	AT&T Administrator's Reference Manual
paste(1)	AT&T User's Reference Manual
pause(2)	AT&T Programmer's Reference Manual
pcat(1)	See pack(1)
pclose(3S)	See popen(3S)
pdp11(1)	See machid(1
perror(3C)	AT&T Programmer's Reference Manual
pg(1)	AT&T User's Reference Manual
pipe(2)	AT&T Programmer's Reference Manual
plock(2)	AT&T Programmer's Reference Manual
pnch(4)	AT&T Administrator's Reference Manual
poll(2)	AT&T Programmer's Reference Manual
popen(3S)	AT&T Programmer's Reference Manual
pow(3M)	See exp(3M)
powerfail(1M)	D-NIX 5.3 Reference Manual
pr(1)	D-NIX 5.3 Reference Manual
prctmp(1M)	See acctsh(1M)
prdaily(1M)	AT&T Administrator's Reference Manual
print(1)	D-NIX 5.3 Reference Manual
printf(3S)	AT&T Programmer's Reference Manual
prof(1)	AT&T Programmer's Reference Manual
prof(5)	AT&T Administrator's Reference Manual
profil(2)	AT&T Programmer's Reference Manual
profile(4)	AT&T Administrator's Reference Manual
prs(1)	AT&T Programmer's Reference Manual
prtacct(1M)	See acctsh(1M)
ps(1)	D-NIX 5.3 Reference Manual
ptrace(2)	AT&T Programmer's Reference Manual
ptx(1)	AT&T User's Reference Manual
putc(3S)	AT&T Programmer's Reference Manual
putchar(3S)	See putc(3S)
putenv(3C)	AT&T Programmer's Reference Manual
putmsg(2)	AT&T Programmer's Reference Manual
putpwent(3C)	AT&T Programmer's Reference Manual
puts(3S)	AT&T Programmer's Reference Manual
putspent(3X)	AT&T Programmer's Reference Manual
pututline(3C)	See getut(3C)
putw(3S)	See putc(3S)
pwck(1M)	AT&T Administrator's Reference Manual
pwconv(1M)	D-NIX 5.3 Reference Manual
pwd(1)	D-NIX 5.3 Reference Manual
pwunconv(1M)	D-NIX 5.3 Reference Manual

Q

qsort(3C)	AT&T Programmer's Reference Manual
queue(1)	D-NIX 5.3 Reference Manual
quiz(6)	AT&T User's Reference Manual

R

rand(3C)	AT&T Programmer's Reference Manual
rcs(1B)	AT&T User's Reference Manual
rcsdiff(1B)	AT&T User's Reference Manual
rcsfile(5B)	AT&T Administrator's Reference Manual
rcsintro(1B)	AT&T User's Reference Manual
rcsmerge(1B)	AT&T User's Reference Manual
read(2)	AT&T Programmer's Reference Manual
readdir(3X)	See directory(3X)
readonly(1)	D-NIX 5.3 Reference Manual
realloc(3C)	See malloc(3C)
realloc(3X)	See malloc(3X)
red(1)	D-NIX 5.3 Reference Manual
regcmp(1)	AT&T Programmer's Reference Manual
regcmp(3X)	AT&T Programmer's Reference Manual
regex(3X)	See regcmp(3X)
regexp(5)	AT&T Administrator's Reference Manual
reject(1M)	D-NIX 5.3 Reference Manual
reloc(4)	AT&T Administrator's Reference Manual
remlp(1)	D-NIX 5.3 Reference Manual
rewind(3S)	See fseek(3S)
rewinddir(3X)	See directory(3X)
rinstall(1M)	D-NIX 5.3 Reference Manual
rlog(1B)	AT&T User's Reference Manual
rm(1)	D-NIX 5.3 Reference Manual
rmail(1)	D-NIX 5.3 Reference Manual
rmdel(1)	AT&T Programmer's Reference Manual
rmdir(1)	See rm(1)
rmdir(2)	AT&T Programmer's Reference Manual
rmuser(1M)	D-NIX 5.3 Reference Manual
rsh(1)	D-NIX 5.3 Reference Manual
runacct(1M)	AT&T Administrator's Reference Manual

S

sact(1)	AT&T Programmer's Reference Manual
sar(1M)	AT&T Administrator's Reference Manual
sbrk(2)	See brk(2)
scanf(3S)	AT&T Programmer's Reference Manual
sccsdiff(1)	AT&T Programmer's Reference Manual

sccsfile(4)	AT&T Administrator's Reference Manual
sccstorcs(8B)	AT&T Administrator's Reference Manual
scnhdr(4)	AT&T Administrator's Reference Manual
scr_dump(4)	AT&T Administrator's Reference Manual
scrfile(1)	D-NIX 5.3 Reference Manual
sdb(1)	AT&T Programmer's Reference Manual
sdiff(1)	AT&T User's Reference Manual
sed(1)	D-NIX 5.3 Reference Manual
seed48(3C)	See drand48(3C)
seekdir(3X)	See directory(3X)
semctl(2)	AT&T Programmer's Reference Manual
semget(2)	AT&T Programmer's Reference Manual
semop(2)	AT&T Programmer's Reference Manual
set(1)	D-NIX 5.3 Reference Manual
setbuf(3S)	AT&T Programmer's Reference Manual
setgid(2)	See setuid(2)
setgrent(3C)	See getgrent(3C)
setjmp(3C)	AT&T Programmer's Reference Manual
setkey(3C)	See crypt(3C)
setmnt(1M)	D-NIX 5.3 Reference Manual
setpgrp(1)	AT&T User's Reference Manual
setpgrp(2)	AT&T Programmer's Reference Manual
setpwent(3C)	See getpwent(3C)
setspeed(1)	D-NIX 5.3 Reference Manual
setuid(2)	AT&T Programmer's Reference Manual
setutent(3C)	See getut(3C)
setvbuf(3S)	See setbuf(3S)
sh(1)	D-NIX 5.3 Reference Manual
shmctl(2)	AT&T Programmer's Reference Manual
shmget(2)	AT&T Programmer's Reference Manual
shmop(2)	AT&T Programmer's Reference Manual
shutacct(1M)	See acctsh(1M)
shutdown(1M)	D-NIX 5.3 Reference Manual
sighold(2)	See sigset(2)
sigigmore(2)	See sigset(2)
signal(2)	AT&T Programmer's Reference Manual
sigpause(2)	See sigset(2)
sigrelse(2)	See sigset(2)
sigset(2)	AT&T Programmer's Reference Manual
sin(3M)	See trig(3M)
sinh(3M)	AT&T Programmer's Reference Manual
siv(1)	D-NIX 5.3 Reference Manual
size(1)	AT&T Programmer's Reference Manual
sleep(1)	D-NIX 5.3 Reference Manual
sleep(3C)	AT&T Programmer's Reference Manual
sno(1)	AT&T User's Reference Manual
sort(1)	D-NIX 5.3 Reference Manual
spell(1)	AT&T User's Reference Manual
spellin(1)	See spell(1)
spline(1G)	AT&T User's Reference Manual
split(1)	AT&T User's Reference Manual
sprintf(3S)	See printf(3S)

sqrt(3M)	See exp(3M)
srand(3C)	See rand(3C)
srand48(3C)	See drand48(3C)
sscanf(3S)	See scanf(3S)
ssignal(3C)	AT&T Programmer's Reference Manual
starter(1)	AT&T User's Reference Manual
startup(1M)	See acctsh(1M)
stat(2)	AT&T Programmer's Reference Manual
stat(5)	AT&T Administrator's Reference Manual
statfs(2)	AT&T Programmer's Reference Manual
stdio(3S)	AT&T Programmer's Reference Manual
stdipc(3C)	AT&T Programmer's Reference Manual
stime(2)	AT&T Programmer's Reference Manual
strcat(3C)	See string(3C)
str...(3C)	See string(3C)
streamio(7)	AT&T Administrator's Reference Manual
string(3C)	AT&T Programmer's Reference Manual
strip(1)	AT&T Programmer's Reference Manual
strtod(3C)	AT&T Programmer's Reference Manual
strtol(3C)	AT&T Programmer's Reference Manual
stty(1)	D-NIX 5.3 Reference Manual
su(1M)	D-NIX 5.3 Reference Manual
sum(1)	AT&T User's Reference Manual
swab(3C)	AT&T Programmer's Reference Manual
syms(4)	AT&T Administrator's Reference Manual
sync(1M)	D-NIX 5.3 Reference Manual
sync(2)	AT&T Programmer's Reference Manual
sys_errlist(3C)	See perror(3C)
sys_nerr(3C)	See perror(3C)
sysfs(2)	AT&T Programmer's Reference Manual
system(3S)	AT&T Programmer's Reference Manual

T

tabs(1)	AT&T User's Reference Manual
tail(1)	AT&T User's Reference Manual
tan(3M)	See trig(3M)
tanh(3M)	See sinh(3M)
tar(1)	D-NIX 5.3 Reference Manual
tbl(1)	AT&T User's Reference Manual
tc(1M)	D-NIX 5.3 Reference Manual
tdelete(3C)	See tsearch(3C)
tee(1)	AT&T User's Reference Manual
telinit(1M)	D-NIX 5.3 Reference Manual
telldir(3X)	See directory(3X)
tempnam(3S)	See tmpnam(3S)
term(4)	AT&T Administrator's Reference Manual
term(5)	AT&T Administrator's Reference Manual
terminfo(4)	AT&T Administrator's Reference Manual

termio(7)	AT&T Administrator's Reference Manual
test(1)	D-NIX 5.3 Reference Manual
tfind(3C)	See tsearch(3C)
tic(1M)	AT&T Programmer's Reference Manual
time(1)	D-NIX 5.3 Reference Manual
time(2)	AT&T Programmer's Reference Manual
times(1)	D-NIX 5.3 Reference Manual
times(2)	AT&T Programmer's Reference Manual
timezone(4)	AT&T Administrator's Reference Manual
tmpfile(3S)	AT&T Programmer's Reference Manual
tmpnam(3S)	AT&T Programmer's Reference Manual
toascii(3C)	See conv(3C)
tolower(3C)	See conv(3C)
touch(1)	D-NIX 5.3 Reference Manual
toupper(3C)	See conv(3C)
tput(1)	AT&T User's Reference Manual
tr(1)	D-NIX 5.3 Reference Manual
trenter(1M)	AT&T Administrator's Reference Manual
trig(3M)	AT&T Programmer's Reference Manual
true(1)	D-NIX 5.3 Reference Manual
tsearch(3C)	AT&T Programmer's Reference Manual
tsort(1)	AT&T Programmer's Reference Manual
ttt(6)	AT&T User's Reference Manual
tty(1)	D-NIX 5.3 Reference Manual
tty(7)	AT&T Administrator's Reference Manual
ttyname(3C)	AT&T Programmer's Reference Manual
ttyslot(3C)	AT&T Programmer's Reference Manual
turnacct(1M)	See acctsh(1M)
twalk(3C)	See tsearch(3C)
type(1)	D-NIX 5.3 Reference Manual
types(5)	AT&T Administrator's Reference Manual
tzset(3C)	See ctime(3C)

U

ulimit(2)	AT&T Programmer's Reference Manual
umask(1)	D-NIX 5.3 Reference Manual
umask(2)	AT&T Programmer's Reference Manual
umount(1)	D-NIX 5.3 Reference Manual
umount(2)	AT&T Programmer's Reference Manual
uname(1)	D-NIX 5.3 Reference Manual
uname(2)	AT&T Programmer's Reference Manual
unget(1)	AT&T Programmer's Reference Manual
ungetc(3S)	AT&T Programmer's Reference Manual
uniq(1)	D-NIX 5.3 Reference Manual
unistd(4)	AT&T Administrator's Reference Manual
units(1)	AT&T User's Reference Manual
unlink(1M)	See link(1M)

unlink(2)	AT&T Programmer's Reference Manual
unpack(1)	See pack(1)
unset(1)	D-NIX 5.3 Reference Manual
usage(1)	AT&T User's Reference Manual
ustat(2)	AT&T Programmer's Reference Manual
utime(2)	AT&T Programmer's Reference Manual
utmp(4)	AT&T Administrator's Reference Manual
utmpname(3C)	See getut(3C)
uucheck(1M)	AT&T Administrator's Reference Manual
uucico(1M)	AT&T Administrator's Reference Manual
uucleanup(1M)	AT&T Administrator's Reference Manual
uucp(1C)	AT&T User's Reference Manual
uulog(1C)	See uucp(1)
uuname(1C)	See uucp(1)
uupick(1C)	See uuto(1)
uusched(1M)	AT&T Administrator's Reference Manual
uustat(1C)	AT&T User's Reference Manual
uuto(1C)	AT&T User's Reference Manual
Uutry(1M)	AT&T Administrator's Reference Manual
uux(1C)	AT&T User's Reference Manual
uuxqt(1M)	AT&T Administrator's Reference Manual

V

val(1)	AT&T Programmer's Reference Manual
values(5)	AT&T Administrator's Reference Manual
varargs(5)	AT&T Administrator's Reference Manual
vax(1)	See machid(1)
vc(1)	AT&T Programmer's Reference Manual
vi(1)	AT&T User's Reference Manual
vfprintf(3S)	See vprintf(3S)
vprintf(3S)	AT&T Programmer's Reference Manual
vprintf(3X)	AT&T Programmer's Reference Manual
vsar(1)	AT&T User's Reference Manual
vsprintf(3S)	See vprintf(3S)

W

wait(1)	D-NIX 5.3 Reference Manual
wait(2)	AT&T Programmer's Reference Manual
wall(1)	D-NIX 5.3 Reference Manual
wc(1)	D-NIX 5.3 Reference Manual
what(1)	AT&T Programmer's Reference Manual
who(1)	D-NIX 5.3 Reference Manual
whodo(1M)	AT&T Administrator's Reference Manual
write(1)	D-NIX 5.3 Reference Manual

write(2)	AT&T Programmer's Reference Manual
wtmp(4)	See utmp(4)
wtmpfix(1M)	See fwtmp(1M)
wump(6)	AT&T User's Reference Manual

X

xargs(1)	AT&T User's Reference Manual

Y

yacc(1)	AT&T Programmer's Reference Manual
y*n*(3M)	See bessel(3M)