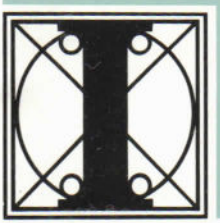
LSX Computer Line (Up to Model 3040)

Operating Systems

X/OS UNIX® System V-based Operating System Utilities

Reference Manual

X/OS



olivetti

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 Ivrea (Italy)

Information from
Olivetti Documentation

LSX Computer Line (Up to Model 3040)

Operating Systems

X/OS UNIX® System V-based Operating System Utilities

Reference Manual

olivetti

PREFACE

This is a reference manual for the general user and describes the X/OS (UNIX System V-based operating system) commands and utilities.

SUMMARY

This manual is divided into:

- the list of contents
- a permuted index
- the command and utility descriptions in alphabetically ordered manual pages

REFERENCES

Read first ...

LSX X/OS Operating Guide - Code 4055390 Y

X/OS User Manual - Code 4043610 C

X/OS Advanced Utilities User Guide - Code 4043620 D

DISTRIBUTION: As part of Software Kit (W)

FIRST EDITION: December 1987 - X/OS Release 1.0

This manual describes the X/OS (UNIX System V-based operating system) commands and utilities. Each description takes the form of a formal reference to one or more commands. All of the commands described in this manual are used directly by the general user.

The number appended to the title of each command page acts as a guide to the type of facility described. Most of the pages in this manual are assigned the number 1, which indicates the general commands and utilities. Some have the number 1C, which refers to the general communications systems.

Some descriptions include more than one command. For example, the pages describing the **at** command also cover **batch**. To find the reference for **batch**, for example, see the Permuted Index or the alphabetical list of commands.

Each reference uses the same format, which is summarised below. Note that not all of the following sections are always present for each command.

NAME — gives the name or names of the functions described. A brief one-line description is given to indicate the purpose of the command.

SYNOPSIS — summarises the format of the command, giving mandatory elements and options. The conventions used are as follows:

- **Boldface** text represents literal strings, and should be typed exactly as shown.
- Normal text identifies substitutable arguments. For example, the word "file" or "name" indicates that a filename should be given.
- *Italic* text, if used, has the same meaning as normal text.
- [ ] Square brackets indicate optional argument.

...	Ellipsis indicate that the previous argument may be repeated.
DESCRIPTION	discusses the features of the command or utility.
EXAMPLES	gives one or more examples of the command or utility.
FILES	lists the filenames that are relevant to the command or utility.
SEE ALSO	gives references to related material.
DIAGNOSTICS	illustrates the diagnostic systems built in to the command or utility. Messages that are self-explanatory are not listed.
WARNINGS	points to possible problem areas.
BUGS	indicates some restrictions in the program. A bypass may be indicated.

TABLE OF CONTENTS

The X/OS system provides all of the X/OPEN standard services. It also provides various additional services. The services that conform to the X/OPEN standard are marked with an asterisk in the table of contents.

NAME

intro - introduction to commands and application programs

DESCRIPTION

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

(1) Commands of general utility.

(1C) Commands for communication with other systems.

COMMAND SYNTAX

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

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

where:

name The name of an executable file.

option can be:

noargletter(s) or

argletter<>optarg

where

noargletter A single letter representing an option without an argument.

argletter A single letter representing an option requiring an argument.

<> an optional white space

optarg An argument (character string) satisfying preceding *argletter*.

cmdarg A pathname (or other command argument) *not* beginning with - or - by itself indicating the standard input.

SEE ALSO

getopt(1), getopt(3C).

DIAGNOSTICS

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

NAME

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

SYNOPSIS

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

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

DESCRIPTION

300 supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; *300s* performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5% to 70%. *300* can be used to print equations neatly, in the sequence:

neqn *file* ... | **nroff** | **300**

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

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

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

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

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

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

300 can be used with the *nroff* **-s** flag or **.rd** requests, when it is

necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the carriage return key in these cases, you must use the line-feed key to get any response.

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

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

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

The *neqn* names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(5).

SEE ALSO

450(1), eqn(1), graph(1G), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).
Advanced Utilities User Guide.

BUGS

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

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

NAME

4014 - paginator for the Tektronix 4014 terminal

SYNOPSIS

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

DESCRIPTION

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

The command line options are:

-t Don't wait between pages (useful for directing output into a file).

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

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

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

SEE ALSO

pr(1), tc(1), troff(1).
Advanced Utilities User Guide.

BUGS

When *4014* is invoked with the **-t** flag, the process does not terminate if the line drops.

NAME

450 - handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

450 supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the DIABLO 1620 or XEROX 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as *300*(1). *450* can be used to print equations neatly, in the sequence:

neqn *file ...* **| nroff | 450**

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

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

In many (but not all) cases, the use of *450* can be replaced by one of the following:

nroff -T450 *files ...*

or

nroff -T450-12 *files* ...

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

The *neqn* names of, and resulting output for, the Greek and special characters supported by *450* are shown in *greek*(5).

SEE ALSO

300(1), eqn(1), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), greek(5).

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there. If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

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/MD** (total blocks read and written).

The command name is prepended with a **#** if it was executed with superuser 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).

-q Do not print any output records; just print the average statistics, as with the **-a** option.

-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 superuser privileges, or **?** which designates only those processes associated with unknown user IDs.

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

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

-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.

-o *ofile* Copy selected process records in the input data format to *ofile*; suppress 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*.

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

FILES

/etc/passwd

/usr/adm/pacct
/etc/group

SEE ALSO

ps(1), su(1), acct(2), acct(4), utmp(4). acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time and option **-d** is not used, then *time* is interpreted as occurring on the previous day.

NAME

asa - interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

Asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no filenames are supplied. The first character of each line is assumed to be a control character; their meanings are:

' ' (blank) single new line before printing

0 double new line before printing

1 new page before printing

+ overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ' '. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic will appear on standard error. This program forces the first line of each input file to start on a new page.

To correctly view the output of FORTRAN programs which use ASA carriage control characters, *asa* can be used as a filter:

a.out | asa | lpr

The output, properly formatted and paginated, is directed to the line printer. FORTRAN output sent to a file can be viewed by:

asa *file*

SEE ALSO

ef1(1), f77(1), fsplit(1), ratfor(1) in the *Common Development Tools Reference Manual.*

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. The user can specify when the *at* commands should be executed, while jobs queued with *batch* will execute when system load level permits. The **-r** option can be used to remove a *job* previously scheduled with *at*. The **-l** option reports all jobs scheduled for the invoking user.

Standard output and standard error output are mailed to the user unless they are redirected. 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 either file is **at.deny**, global usage is permitted. The allow/deny files consist of one user name per line.

The *time* may be specified as 1, 2, or 4 digits. One- and two-digit numbers are taken to be hours; four-digit numbers are taken to be

hours and minutes. The time also may 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.)

Examples of valid commands are:

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. Although it is similar to the command **at now**, *batch* goes into a different queue and **at now** will respond with the error message too late.

The command **at -r** removes jobs previously scheduled by *at* or *batch*. The job number is the number given previously by the *at* or *batch* command. Job numbers can also be gotten by using the command **at -l**. Except for the superuser, each user can only remove his or her own jobs.

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 the user's commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
nroff filename > outfile
CTRL-d (hold down the key marked CTRL and press 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 <<!
nroff 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 uses
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

kill(1), mail(1), nice(1), ps(1), sh(1), cron(1M).
Advanced Utilities User Guide

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [**-F***c*] [**-f***prog*] [*parameters*] [*files*]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. The set of patterns may appear literally as *prog*, or they may be specified in a file, **"progfile"**, and executed by the command

awk -f progfile *files*

If the patterns appear literally, the *prog* string(s) should be enclosed in single quotes (') to protect them from the shell. *Parameters*, in the form **x=... y=...** etc., may be passed to *awk*. The examples provided at the end of this description include a command line containing a *parameter*.

Files are read in order; if there are no files, the standard input is read. The filename **-** means the standard input.

The input to *awk* is divided into "records" terminated by a record separator. The default record separator is a new line, so by default *awk* processes its input a line at a time. The input record separator may be changed by defining the variable **RS**.

Each input record is considered to be divided into "fields". The default field separator is white space (blanks or tabs). This separator may be changed by setting the variable **FS** to the character *c* or by using the optional command-line argument **-F***c*. In a pattern statement, the fields are referred to as **$1**, **$2**, ...; **$0** refers to the entire line. If the record separator is empty, an

empty input line is taken as the record separator, and blanks, tabs, and new lines are treated as field separators.

For each pattern in *prog*, an action can be specified. The specified action will be performed on each line or fields within lines that match the pattern.

A pattern-action statement has the form:

pattern { action }

Each line of input is matched against each of the patterns in succession. For each pattern that matches, the associated action is executed. When all the patterns have been tested, the next line of input is fetched and the matching process starts over. Either the pattern or the action may be omitted, but not both. If there is no action for a pattern, the matching line is simply copied to the output. (A line that matches several patterns can be printed several times.) If there is no pattern for an action, then the action is performed for every input line. A line which does not match a pattern is ignored.

An action is a sequence of statements. Since patterns or actions may be omitted, actions must be enclosed in braces to distinguish them from patterns in the program. A statement can be one of the following:

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

Statements are terminated by semicolons, new lines, or right

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

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

The built-in function *length* returns the length of its argument taken as a string, or the length of the whole line if no argument is given. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. *Int* 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 -Fc 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**).

Additional information about patterns and actions and the overall operation of *awk* is provided in the *User Guide*.

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 }
```

A command line using parameters:

```
awk -f program n=5 input
```

SEE ALSO

grep(1), lex(1), sed(1).
User 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

banner - make posters

SYNOPSIS

banner *strings*

DESCRIPTION

Banner prints its arguments in large letters on the standard output. Each argument can be up to 10 characters long, not counting spaces. *Banner* can be used to create a file for later printing with the command

banner *strings* **>** *file*

or the *banner* command can be piped directly to a line printer with the command line

banner *strings* **| lp**

SEE ALSO

echo(1).

NAME

basename, dirname - deliver portions of pathnames

SYNOPSIS

basename *string* [*suffix*]

dirname *string*

DESCRIPTION

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

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

EXAMPLES

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

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

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

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

SEE ALSO

sh(1).

BUGS

The *basename* of / is null and is considered an error.

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

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

DESCRIPTION

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

Comments
    are enclosed in /* and */.

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

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

Operators
    + - * / % ^ (% is remainder; ^ is power)
    ++ -- (prefix and postfix; apply to names)

```
        == <= >= != < >
        = =+ =- =* =/ =% =

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

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

    Functions in -l math library
        s(x) sine
        c(x) cosine
        e(x) exponential
        l(x) log
        a(x) arctangent
        j(n,x)    Bessel function
```

All function arguments are passed by value.

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

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

variables empty square brackets must follow the array name.

Bc is actually a preprocessor for *dc*(1), which it invokes automatically, unless the **-c** (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

EXAMPLE

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

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

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

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

FILES

/usr/lib/lib.b mathematical library

/usr/bin/dc desk calculator proper

SEE ALSO

dc(1).

Advanced Utilities User Guide.

BUGS

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

NAME

bdiff - file comparator for large files

SYNOPSIS

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

DESCRIPTION

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

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

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

bfs - big file scanner

SYNOPSIS

bfs [-] *name*

DESCRIPTION

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

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

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?. The symbol > indicates downward search without wrap-around; < 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**, **n**, **p**, **q**, **w**, **=**, **!** and null commands operate as described under *ed*. Commands such as **---**, **+++-**, **+++=**, **-12**, and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being

scanned; there is no "remembered" filename. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt**, and **xc** commands, below). The following additional commands are available:

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

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. 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:

  Either address is not between 1 and $.

  The second address is less than the first.

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

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

xb/^/ label

is an unconditional jump.

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

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

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

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

1,%5p
1,%5
%6

The command

g/%5/p

produces a global search for the characters **100** and prints each line containing a match. To escape the special meaning of **%**, a **** must precede it.

g/".*[cds]/p

can 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 an operating system command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example, the lines

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

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 ****.

The command

```
xv7\!date
```

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

xbz *label*

xbn *label*

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

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

```
: l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz 1
```

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

SEE ALSO

csplit(1), ed(1), regcmp(3X).
Advanced Utilities User Guide

DIAGNOSTICS

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

NAME

cal - print calendar

SYNOPSIS

cal [[*month*] *year*]

DESCRIPTION

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

Try September 1752.

BUGS

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

Beware that "cal 87" refers to the early Christian era, not the 20th century.

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

Calendar consults the file **calendar** in a user's current directory and prints out lines containing today's or tomorrow's date. *Calendar* uses *calprog* to figure out today's and tomorrow's dates. The date read by *calprog* may appear anywhere in a line, and most reasonable date representations are recognized, although the month must appear first. For example, "Dec. 7," "december 7," and "12/7" are recognized; "7 Dec" or "seven december" are not. On weekends "tomorrow" extends through Monday. If the *calendar* command is run on a Friday, lines containing the dates for Friday, Saturday, Sunday, and Monday are selected.

When an argument is present, *calendar* does its job for all users who have a file **calendar** in their login directory and sends them any positive results by *mail*(1). Normally this is done daily by facilities in the operating system.

FILES

calendar
/usr/lib/calprog
/etc/passwd
/tmp/cal*

SEE ALSO

mail(1).

BUGS

The **calendar** file must be public information for a user to get reminder service.

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

Numeric dates must be in the form month/day. Separators other than a slash prevent recognition of the date.

NAME

cat - concatenate and print files

SYNOPSIS

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

DESCRIPTION

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

cat file

prints the file, and:

cat file1 file2 > file3

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

If no input file is given, or if the argument **-** is encountered, *cat* reads from the standard input file. Output is buffered unless the **-u** option is specified. The **-s** option makes *cat* silent about non-existent files.

The **-v** option causes non-printing characters (with the exception of tab, new-line, and form-feed characters) to be printed visibly. Control characters are printed **^X**, **CTRL-x**; the **DEL** character (octal 0177) is printed **^?**. Non-ASCII characters (with the high bit set) are printed as **M-**x, where *x* is the character specified by the seven low- order bits.

When used with the **-v** option, **-t** causes tabs to be printed as **^I** and **-e** causes a **$** character to be printed at the end of each line (prior to the new-line character). The **-t** and **-e** options are

ignored if the -v option is not specified.

WARNING

Command formats such as

cat file1 file2 > file1

cause the original data in **file1** to be lost. To append **file2** to **file1**, use:

cat file2 >> file1.

SEE ALSO

cp(1), pg(1), pr(1).

NAME

cb - C program beautifier

SYNOPSIS

cb [**-s**] [**-j**] [**-l** *leng*] [*file* ...]

DESCRIPTION

Cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that display the structure of the code. Under default options, *cb* preserves all user new- lines. Under the **-s** flag, *cb* canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The **-j** flag causes split lines to be put back together. The **-l** flag causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).
The C Programming Language by B. W. Kernighan and D. M. Ritchie.

BUGS

Punctuation that is hidden in preprocessor statements causes indentation errors.

NAME

cd - change working directory

SYNOPSIS

cd [*directory*]

DESCRIPTION

The *cd* command is used to change from one working directory to another. The optional argument *directory* can be specified in several ways. A complete pathname can be specified (e.g., **cd /usr/food/cabbage/cooked**). *Directory* can also specify a pathname relative to the current directory. If the current directory is **/usr** and you want to change to **/usr/food**, the command would be **cd food**. If *directory* is not specified, the value of shell parameter **$HOME** is used as the new working directory. If the command **cd ..** is given, the new working directory will be one level higher in the directory hierarchy. For example, if the current working directory is **/x/y/z**, and the command **cd ..** is given, the new working directory will be **/x/y**. "Search paths" may be specified by the shell variables **$PATH** and **$CDPATH** (refer to *sh*(1)). The current values for these variables can be reviewed by using the *env*(1) command.

Cd must have execute (search) permission in *directory*.

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

SEE ALSO

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

NAME

chmod - change mode

SYNOPSIS

chmod *mode files*

DESCRIPTION

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

04000	set user ID on execution
02000	set group ID on execution
00400	read by owner
00200	write by owner
00100	execute (search in directory) by owner
00040	read by group
00020	write by group
00010	execute (search in directory) by group
00004	read by others
00002	write by others
00001	execute (search in directory) by others

For example, to obtain the absolute mode to give read and write permissions to the owner and read permission to the group and others, add 0400, 0200, 0040, and 0004. The command to change the permissions on the file would be

chmod 644 file

A symbolic *mode* has the form:

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

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

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

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

If more than one change is to be made to the permissions of a file using symbolic mode, there cannot be any spaces in the mode argument. Multiple changes are separated by commas. For example

chmod u+w,go+x file

gives write permission to the user and execute permission to group and others. Operations are performed in the order specified. The letter **s** is useful only with **u** or **g**.

Only the owner of a file (or the superuser) may change its mode. Use **ls -l** *file* to check permissions on a file or to verify permission changes.

In order to set the group ID, the group of the file must correspond to the user's current group ID.

EXAMPLES

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

chmod o-w file

chmod +x file

SEE ALSO

ls(1), chmod(2).

DIAGNOSTICS

chmod: can't access filename
Named file or directory does not exist.

chmod: can't change filename
User does not own named file.

chmod: invalid mode
Specified mode is not in correct absolute or symbolic format.

NAME

chown, chgrp - change owner or group

SYNOPSIS

chown *owner file ...*

chgrp *group file ...*

DESCRIPTION

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

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

FILES

/etc/passwd

/etc/group

SEE ALSO

chown(2), group(4), passwd(4).

NAME

clear - clear terminal screen

SYNOPSIS

clear

DESCRIPTION

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

FILES

/etc/termcap terminal capability data base

NAME

cmp - compare two files

SYNOPSIS

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

DESCRIPTION

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

Options:

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

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

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

Exit codes returned are: 0 for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

col - filter reverse line feeds

SYNOPSIS

col [**-bfpx**]

DESCRIPTION

Col reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code **ESC-7**), and by forward and reverse half-line feeds (**ESC-9** and **ESC-8**). *Col* is particularly useful for filtering multicolumn output made with the **.rt** command of *nroff* 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 is 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 never contains either kind of reverse line motion.

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

The ASCII control characters Shift Out (**SO**) (\017) and Shift In (**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* ignores any unknown escape sequences found in the 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.

EXAMPLE

The command *col* is typically used together with *nroff*(1) and *tbl*(1) as shown:

```
tbl doc.d | nroff -mm | col > doc.o
```

In this command line, *doc.d* and *doc.o* are source document and formatted document respectively. If *col* is not used, it will be uncertain whether the output is printable. The following is an example of the output produced without *col*:

```
^[8     _____________________________________________________

        NAME                                          CITY

^[8     _____________________________________________________

        Brown
^[7                                                   New York

        Green
^[7                                                   Los Angeles

        White
```

^[7 Philadelphia

^[8 ———————————————————————————————

^[9

Note that some printers will not interpret the special characters **^[8**, **^[9** and **^[7**, so *col* must be used to obtain:

NAME	CITY
Brown	New York
Green	Los Angeles
White	Philadelphia

SEE ALSO

nroff(1), tbl(1).

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* is a device that can interpret half-line motions; otherwise, use **-Tlp**.

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 filename - means the standard input.

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

SEE ALSO

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

NAME

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

SYNOPSIS

cp *file1* [*file2* ...] *target*

ln [**-f**] [**-s**] *filename* [*linkname*]

mv [**-f**] *file1* [*file2* ...] *target*

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same (take care when using *sh*(1) metacharacters). If *target* is a directory, then one or more files are copied (linked, moved) to that directory. If *target* is a file, its contents are destroyed.

If *mv* determines that the mode of *target* forbids writing, it prints the mode (see *chmod*(2)), asks for a response, and reads the standard input for one line (if the standard input is a terminal); if the line begins with **y**, the move or link takes place; if not, the command exits. No questions are asked and the *mv* or *ln* is done when the **-f** option is used or when the standard input is not a terminal.

Only *mv* allows *file1* to be a directory, in which case the directory rename occurs only if the two directories have the same parent; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

When using *cp*, if *target* is not a file, a new file is created with the same mode as *file1* except that the sticky bit is not set unless the *cp* is executed by the superuser; the owner and group of *target*

are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, or group. The last modification time of *target* (and last access time if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

Ln assigns an additional name (directory entry), called a *link* for a file or directory. A hard link, which is the default, can only be made to an existing file. To remove a file with more than one hard link, all such links (including the name by which it was created) must be removed. Hard links may not span file systems.

A symbolic link is specified using the **-s** option. A symbolic link, which may span file systems, can be made to a filename, or to a directory, whether or not the named file exists. A symbolic link does not prevent the original file from being removed.

SEE ALSO

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

BUGS

If *file1* and *target* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Ln does not link across file systems.

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o [**acBv**]

cpio -i [**BcdmrtuvfsSb6**] [*patterns*]

cpio -p [**adlmruv**] *directory*

DESCRIPTION

Cpio -o (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information.

Cpio -i (copy in) extracts files from the standard input which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1). In *patterns*, metacharacters **?**, *****, and **[...]** match the slash **/** character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is ***** (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

Cpio -p (pass) reads the standard input to obtain a list of pathnames of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

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

B Block input/output 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from **/dev/rmt?**).

d Create *directories* as needed.

c Write *header* information in ASCII character form for portability.

r Interactively *rename* files. If the user types a null line, the file is skipped.

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

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

v *Verbose*: print a list of filenames. When used with the **t** option, the table of contents looks like the output of an **ls -l** command (see *ls*(1)).

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

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

f Copy in all files except those in *patterns*.

s Swap bytes. Use only with the **-i** option.

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

b Swap both bytes and halfwords. Use only with the **-i** option.

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

EXAMPLES

1. The example below copies all the **.c* files in a directory to an archive on a magnetic tape unit:

   ```
   ls *.c | cpio -o >dev/mt0
   ```

   The special file */dev/mt0* is the logical magnetic tape device.

   To restore the contents of the archive whilst renaming files, use:

   ```
   cat /dev/mt0 | cpio -ir
   ```

   Option **r** indicates the renaming of restored files. Use this option if you want restore files into a directory whose name is different from the source directory name.

2. Be careful when restoring files onto a different machine. Some machines may have a different system of byte management, so it is necessary to use byte swapping during restoration. The following command line executes this procedure:

   ```
   cat /dev/mt0 | cpio -is
   ```

   Option **s** swaps bytes. Use option **S** to swap halfwords and **b** to swap both bytes and halfwords.

3. To duplicate directory *olddir* in *newdir*:

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

   The option *depth* causes descent of the directory to be performed in order that all entries in the directory are acted on before the directory itself.

4. The trivial case

   ```
   find . -depth -print | cpio -oB >dev/rmt0
   ```

can be handled more efficiently by:

find . -cpio dev/rmt0

SEE ALSO

find(1), cpio(4).

BUGS

Pathnames are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory; thereafter, linking information is lost. Only the superuser can copy special files.

NAME

crontab - user crontab file

SYNOPSIS

crontab [*file*]

crontab -r

crontab -l

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontab files. The **-r** option removes a user's crontab from the crontab directory. The command **crontab -l** will list the crontab file for the invoking user.

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

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five fields are integer patterns that specify the following:

- minute (0-59)

- hour (0-23)

- day of the month (1-31)

- month of the year (1-12)

- day of the week (0-6 with 0=Sunday)

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, **0 0 1,15 * 1** would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, **0 0 * * 1** would run a command only on Mondays).

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

The shell is invoked from the **$HOME** directory with an **arg0** of **sh.** Users who desire to have their **.profile** executed must explicitly do so in the crontab file. *Cron*(1M) supplies a default environment for every shell, defining **HOME**, **LOGNAME**, **SHELL(=/bin/sh)**, and **PATH(=:/bin:/usr/bin:/usr/lbin)**.

NOTE: Users should remember to redirect the standard output and standard error of their commands. If this is not done, any generated output or errors will be mailed to the user.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area

/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

sh(1), cron(1M).
System Administration Utilities Reference Manual.
Advanced Utilities User Guide

NAME

crypt - encode/decode

SYNOPSIS

crypt [*password*]

DESCRIPTION

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

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

will print the clear.

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

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

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

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. to take a substantial fraction of a second to compute. How ever, if keys

are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(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**) and a C-like syntax.

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 and from **/etc/login**.

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 **&**, **|**, **;**, **<**, **>**, **(** and **)** form separate words. If doubled, as 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 the backslash (\). 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 hash 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 semi-colons (;), 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 parentheses, ( ), to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with || or && indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions*.)

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

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**. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous. It is also possible to 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 exits. It normally informs you whenever a job changes status, 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.

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 provison 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 the characters **^|**. 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 colon (:) and a designator for the desired words. The words of a input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

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

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 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 non-blank character of an input line is a **^|**. This is equivalent to **!:s^|** providing a convenient shorthand for substitutions on the text of the previous line. Thus **^|lb^|***lib* fixes the spelling of

lib in the previous command. Finally, a history substitution may be surrounded with { and } if necessary to insulate it from the characters which follow. Thus, after **ls -ld ~paul** we might do **!{l}a** to do **ls -ld ~paula**, while **!la** would look for a command starting *la*.

Quotations with ' and "

The quotation of strings by ' and " can be used to prevent all or some of the remaining substitutions. Strings enclosed in ' are prevented any further interpretation. Strings enclosed in " 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 * | lp'** 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 and

${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 : modifiers and the other forms given below are not available in this case).

$name[selector]
and

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

$#name and

${#name} give 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 and

${number}** are equivalent to **$argv[*number*].

$*** is equivalent to **$argv[*].

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

The following substitutions may not be modified with : modifiers.

$?name and

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

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

Input/output

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

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

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

as standard input.

> name

>! name

>& name and

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

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

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

>> name

>>& name

>>! name and

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

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 (**&**, **|**, **<**, **>**, **(** and **)**), 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* and

alias *name wordlist*

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

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

break Causes execution to resume after the *end* of the nearest enclosing *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 and

chdir *name* Change the shells working directory to directory *name*. If no argument is given then change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory (and does not begin with /, ./ or ../), then each component of the variable **cdpath** is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with /, then this is tried to see if it is a directory.

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

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

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

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 and

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

foreach *name* (*wordlist*)

...

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

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

glob *wordlist*

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

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

history

history *n*

history -r *n*

and

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; else if *expr2* is true then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The

words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else*.)

jobs and

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* ...
and

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.

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*
and

nice *+number command*

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using **nice -number ...** *Command* is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

nohup and

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 and

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 - and

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 and

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

pushd

pushd *name* and

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 *n*th 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*and

set *name=(wordlist)*

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

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

setenv *name value*

Sets the value of environment variable **name** to be *value*, a single 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 and

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* and

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.

switch (*string***)**

:case

...

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 and

time *command*

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

ulimit and

ulimit *value*

The user's file size limit is displayed (first form) or set to the specified value (second form). The limit is in 512-byte blocks and is inherited by child processes. Files of any size may be read. See also *ulimit*(2).

umask and

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.

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.

@

@ *name* = *expr*and

@ *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 shell variables 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 selectively.

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 **CTRL-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 *exec*(2). Each word in the variable **path** names a directory from which the shell will attempt to execute the command. If it is given neither a **-c** nor a **-t** option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a **-c** or **-t** argument, and in any case for each directory component of **path** which does not begin with a **/**, the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus **(cd ; pwd) ; pwd prints the home** directory; leaving you where you were

(printing this after the home directory), while **cd ; pwd** leaves you in the **home** directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands 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:

-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 shell scripts for the System V shell, *sh*(1), are not compatible with this shell, both *sh* and *csh* will execute *csh* only if the first characters of a script are **#csh**, i.e. if the script starts with a special comment; otherwise, **/bin/sh** will be invoked. In addition, both shells recognize files beginning with **#!**<*path*>, where <*path*> is the full path name of an interpreter for the file (e.g. **#!/bin/sh** would invoke **/bin/sh**). Remaining arguments initialize the variable **argv**.

Signal handling

The shell normally ignores *quit* signals. Jobs running detached (by **&** 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.

FILES

/etc/login	Read by login shell, at login.
~/.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 5120 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.

SEE ALSO

sh(1), access(2), exec(2), fork(2), kill(2), pipe(2), signal(2), umask(2), ulimit(2), wait(2), tty(4), a.out(5), environ(7)

BUGS

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 ;.

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 ... xxn* ;*n* may not be greater than 99. These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*.
01: From the line referenced by *arg1* up to the line referenced by *arg2*.
.
.
.
n+1: From the line referenced by *argn* to the end of *file*.

The options to *csplit* are:

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

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

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

arg1 ... argn

These arguments 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 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 user's responsibility to remove it.

EXAMPLES

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

This example command creates four files, **cobol00** ... **cobol03**. After the split files have been edited, 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 splits 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 is still printed.

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

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

SEE ALSO

ed(1), sh(1), regexp(5).
Advanced Utilities User Guide

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

cu - call another LSX system

SYNOPSIS

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

DESCRIPTION

Cu calls up another LSX system, a terminal, or possibly a non-LSX 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 (110, 150, 300, 600, 1200, 4800, 9600); 300 is the default value. Most modems are either 300 or 1200 baud. For dial-out lines, *cu* chooses a modem speed (300 or 1200) as the slowest available which can handle the specified transmission speed. Directly connected lines may be set to speeds higher than 1200 baud.

-l*line* Specifies a device name to use as the communication line. This can be used to override searching 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 file **/usr/lib/uucp/L-devices**. When the **-l** and **-s** options are sued simultaneously, *cu* will search the **L-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/ttyab); in this case, a telephone number is not required but the string **dir** may be used to specify a null ACU. If the specified device is associated with an auto dialer, a telephone number must be provided.

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

-t Used when dialing an ASCII terminal that has been set to auto-answer. Appropriate mapping of carriage returns to carriage-return-line-feed pairs is set.

-d Causes diagnostic traces to be printed.

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

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

-m Designates a direct line that has modem control.

-n Requests the telephone number to be dialed from the user rather than taking it from the command line.

telno When using an automatic dialer, the argument is the telephone number, with equal signs for secondary dial tone or minus signes for delays, at appropriate places.

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/L.sys** (the appropriate baud rate is also read along with telephone numbers). *Cu* will try each telephone number or direct line for *systemname* in the **L.sys** file until a connection is made or until all the entries are tried.

dir Using **dir** insures that *cu* will use the line specified by the **-l** option. 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:

~. 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; this is 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.

~~ ... Send the line ~ ... to the remote system.

~%break Transmit a BREAK to the remote system.

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

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

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

Data from the remote is diverted (or appended, if >> is used) to *file*. The trailing ~> terminates the diversion.

The use of **~%put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of *echo*(1) and *cat*(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

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 ~~. For example, *uname* can be executed on Z, X, and Y, as follows:

```
uname
Z
~!uname
X
~~!uname
Y
```

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

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud:

cu -s1200 9=2015551212

If the speed is not specified, 300 is the default value.

To log in to a system connected by a direct line:

cu -l /dev/ttyXX dir

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

cu -l /dev/culXX 2015551212

To use a system name:

cu YYYZZZ

FILES

/usr/lib/uucp/L.sys
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uname(1), uucp(1C).
Advanced Utilities User Guide

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

BUGS

Cu buffers input internally.

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

The feature that allows a *uucp* node name to be specified instead of a telephone number will not work in conjunction with the **-l** option, which allows the user to select a specific output line.

If *cu* is run for more than about two hours, the LOCK file (**/usr/spool/uucp/LCK..***) for the output line will get deleted. Another *cu* user will get an error message about not being able to open a device special file if *cu*, thinking that the line is available, attempts to attach itself to the line still in use. The *cu* user who continuously gets such an error message will need to use the **-l** option to select an alternate line (but, as noted above, the ability to call a remote system by name rather than by phone number is lost).

NAME

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

SYNOPSIS

cut -c*list* [*file1 file2 ...*]

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

DESCRIPTION

Use *cut* to remove columns from a table or fields from each line of a file; in data base parlance, *cut* implements the projection of a relation. The fields specified by *list* can be fixed length, i.e., character positions as on a punched card (**-c** option), or the length can vary from line to line and can be marked with a field delimiter character such as *tab* (**-f** option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

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

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

-f*list* The *list* following **-f** (no space) is a list of fields assumed to be separated in the file by a delimiter character (see **-d**); e.g. , **-f1,7** copies the first and seventh field only. Lines with no field delimiters are

passed through intact (useful for table subheadings), unless **-s** is specified.

-d*char* The character following **-d** (no space) is the field delimiter (**-f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.

-s Suppresses lines with no delimiter characters in case of **-f** option. Unless specified, lines with no delimiters are passed through untouched.

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

HINTS

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLES

The command

```
cut -d: -f1,5 /etc/passwd
```

maps user IDs to names.

The command

```
name=`who am i | cut -f1 -d ' '`
```

sets **name** to the current login name.

SEE ALSO

grep(1), paste(1).

DIAGNOSTICS

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

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

no fields The *list* is empty.

NAME

date - print and set the date

SYNOPSIS

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

DESCRIPTION

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

date 10080045

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

If the argument begins with **+**, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by **%** and is replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year - 01 to 12
d	day of month - 01 to 31
y	last 2 digits of year - 00 to 99
D	date as mm/dd/yy
H	hour - 00 to 23
M	minute - 00 to 59
S	second - 00 to 59
T	time as HH:MM:SS
j	day of year - 001 to 366
w	day of week - Sunday = 0
a	abbreviated weekday - Sun to Sat
h	abbreviated month - Jan to Dec
r	time in AM/PM notation

EXAMPLE

The command

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

generates as output:

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

DIAGNOSTICS

No permission you aren't the superuser and are trying to change the date;

bad conversion the date set is syntactically incorrect;

bad format character
the field descriptor is not recognizable.

FILES

/dev/kmem

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dc - desk calculator

SYNOPSIS

dc [*file*]

DESCRIPTION

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

Bc(1), a preprocessor for *dc*, provides infix notation and a C-like syntax to implement functions and reasonable control structures for programs.

The following constructions are recognized by *dc*:

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

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

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

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

d The top value on the stack is duplicated.

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

f All values on the stack are printed.

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

x The top element of the stack is treated as a character string and is executed as a string of *dc* commands.

X The number on the top of the stack is replaced with its scale factor.

[...**]** The bracketed ASCII string is put 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 The top element on the stack is replaced by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! The rest of the line is interpreted as a LSX 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 The output base is pushed 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 The number on the top of the stack is replaced 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!**:

```
[lal+dsa*pla10>y]sy
0sal
lyx
```

SEE ALSO

bc(1).
Advanced Utilities User Guide.

DIAGNOSTICS

x is unimplemented	*x* is an octal number.
stack empty	There are not enough elements on the stack to do what was asked.
Out of space	The free list is exhausted (too many digits).
Out of headers	Too many numbers are being kept.
Out of pushdown	Too many items are on the stack.
Nesting Depth	There are too many levels of nested execution.

NAME

dd - convert and copy a file

SYNOPSIS

dd [*option=value*] ...

DESCRIPTION

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

if=*file*	input filename; standard input is default
of=*file*	output filename; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
seek=*n*	seek *n* records from beginning of output file before copying

count=*n* copy only *n* input records

conv=ascii convert EBCDIC to ASCII

conv=ebcdic convert ASCII to EBCDIC

conv=ibm slightly different map of ASCII to EBCDIC

conv=lcase map alphabetics to lower case

conv=ucase map alphabetics to upper case

conv=swab swap every pair of bytes

conv=noerror do not stop processing on an error

conv=sync pad every input record to *ibs*

conv=...,... several comma-separated conversions

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

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

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

EXAMPLE

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

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

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

SEE ALSO

cp(1).

DIAGNOSTICS

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

BUGS

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

New-line characters are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

deflib - set up shared library files

SYNOPSIS

deflib [*description-file*]

DESCRIPTION

A library description tool called *deflib* automatically generates a set of C language source files necessary to build the shared module and the archive of stubs.

The *description-file* is an optional argument that defines relevant information for the shared library being created. In case this parameter is missing, the input is taken from the standard input. The result of the execution of the command is the creation of three output files within the current directory. The names and the contents of such files are described below.

The format of the description file is defined in a BNF-like notation as follows:

<*library-description*>:
    <*lib-sec*> <*code-sec*> <*path-sec*> <*entry-sec*>

<*lib-sec*>:
    `*LIBRARY*' `:' <*string*>

<*code-sec*>:
    `*CODE*' `:' <*numeric-string*>

<*path-sec*>:
    `*PATHNAME*' `:' <*string*>

<*entry-sec*>:

<entry> | *<entry>* *<entry-sec>*

<entry>:
`*ENTRY*' `:' *<func-list>*

<func-list>:
<string> | *<string>* *<func-list>*

<string>:
any sequence of printable ASCII characters

<numeric-string>:
any sequence of ASCII decimal digits

Note that comments, starting with a # end when a newline character is reached. They can be freely interspersed within the file. Blank lines can be placed anywhere in the file. The format of the various sections is free, in the sense they need not be contained entirely within one line. The order in which the various sections appear is rigidly specified, as above. No section can be omitted.

The various sections provide the following information:

LIBRARY — This section specifies the name of the library. Such a name is used to build the names of the output files (placed in the current directory). Only one instance of this section is allowed.

CODE — This section defines the unique code assigned to the library. The numeric code is a string evaluating to a decimal integer, no larger than 65535. Only one instance of this section is allowed.

PATHNAME — This section specifies a pathname where the shared module can be found in the UNIX file system, for the purpose of making the library available to its client programs. Initially an attempt will be performed to find the numeric code assigned to the library within **/etc/shlib**. In case it is found, an attempt to make the corresponding shared module available will be

performed. Should this fail, or should the numeric code be unavailable within **/etc/shlib**, the pathname specified through this section will be tried instead. NOTE: this allows library developers to test their code, without making it available to the rest of the system. Only one instance of this section is allowed.

ENTRY This section is used to specify the names of the entry points in the library. Each library cannot have more than 65536 entry points. One such section can list all of the entry points. Alternatively, multiple ENTRY sections can be specified.

The output of *deflib* consists of three files whose names are formed by concatenating the name of the library (as discussed above) with the three strings: *tab.c*, *var.c* and *.h*. The first two files must be separately compiled. The first one must be linked to the shared module (the linkage module containing all the shared code). The second one will be part of the archive of stubs available to the linkage editor. The third file is an include file that must be included by all the stubs and defines the unique codes associated to each library entry point.

From the point of view of a library implementor, only the contents of the *.h* file are of relevance. The file contains some declarations of external variables and functions plus a macro definition and a set of constants (each providing the unique code for one library entry point). A comment preceding each constant definition reports the decimal value of the code for the specified entry point, along with an ordered pair representing the library code and the function code within that library.

The mapping between the C identifiers associated to the library entry points and the symbols used for the macro definitions consists of capitalizing all the lower case letters in the original identifier and prefixing the result with two underscore (_) characters. Each function stub shall initialize its pointer to the corresponding shared routine by invoking **SHLINIT()** and specifying to it the appropriate unique code symbol as a parameter.

FILES

/etc/shlib
*lib*var.c
*lib*tab.c
lib.h

SEE ALSO

LSX X/OS Programmer Guide

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 can be used to 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, but incomplete, job. It works only when changed stretches are short and well-separated; however, it does work on files of unlimited length. Options **-e** and **-f** are unavailable with **-h**.

FILES

/tmp/d?????

/usr/lib/diffh for the **-h** option

SEE ALSO

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 (.).

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

Under the **-e** option, *diff3* publishes a script for the editor *ed* that incorporates 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 can be used to 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 . negate the effect of option **-e**.

Diff3 cannot process files longer than 64K bytes.

NAME

diffmk - mark differences between files

SYNOPSIS

diffmk *name1 name2 name3*

DESCRIPTION

Diffmk is a shell procedure that 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 *.

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 can be used to 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.

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** produces a "change mark" on the preceding or following line of output.

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 filenames 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

du - summarize disk usage

SYNOPSIS

du [**-ars**] [*names*]

DESCRIPTION

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

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

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

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

BUGS

If the **-s** option is not used, non-directories given as arguments are not listed.

If there are too many distinct linked files, *du* counts the excess files more than once.

Files with holes in them get an incorrect block count.

NAME

echo - echo arguments

SYNOPSIS

echo [*arg*] ...

DESCRIPTION

Echo writes its arguments on the standard output, separated by blanks and terminated by a new-line character. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b backspace

\c print line without new-line character

\f form feed

\n new line

\r carriage return

\t tab

**** backslash

\x the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *x*, which must start with a zero.

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

SEE ALSO

sh(1).

NAME

ed, red - text editor

SYNOPSIS

ed [-] [**-p** *string*] [*file*]

red [-] [**-p** *string*] [*file*]

DESCRIPTION

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

Red is a restricted version of *ed*. It only allows editing of files in the current directory. It prohibits executing shell commands via !shell *command*. Attempts to bypass these restrictions result in an error message (**restricted shell**).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty -tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops are used automatically when scanning *file*. For example, if the first line of a file contains:

<:t5,10,15 s72:>

Tab stops are set at columns 5, 10 and 15, and a maximum line length of 72 is imposed. NOTE: While inputting text, typed tab characters are expanded to every eighth column, as is the default.

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

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

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

1. The following one-character REs match a *single* character:

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

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

      - The period, asterisk, opening square bracket and backslash (., *, [, and \), which are always special, except when they appear within square brackets ([]; see (d), below).

- ^ (caret or circumflex), which is special at the beginning of an entire RE (see 3(a) and 3(b), below), or when it immediately follows the left of a pair of square brackets ([]) (see (d), below). below).

- $ (currency symbol), which is special at the end of an entire RE (see 3(b), below).

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

c) A period (.) is a one-character RE that matches any character except new line.

d) A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches any single character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character except new line and the remaining characters in the string. The caret (^) has this special meaning only if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, **[0-9]** is equivalent to **[0123456789]**. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after the initial ^, if any); e.g., **[]a-f]** matches either a right square bracket (]) or one of the letters **a** through **f**, inclusive. The four characters listed in the first item of 1(b), above, stand for themselves within such a string of characters.

2. The following rules may be used to construct REs from one-character REs:

a) A one-character RE is a RE that matches whatever the one-character RE matches.

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

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

d) The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

e) A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

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

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

a) A circumflex (^) at the beginning of an entire RE constrains that RE to match an initial segment of a line.

b) A currency symbol ($) at the end of an entire RE constrains that RE to match a final segment of a line.

c) The construction ^*entire RE*$ constrains the entire RE to match the entire line.

d) The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

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

1. The character . addresses the current line.

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

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

4. The sequence 'x addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.

5. An RE enclosed by slashes (/) addresses the first line containing a matching RE found by searching forward from the line following the current line. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.

6. An RE enclosed in question marks (?) addresses the first line containing a matching RE found by searching backward from the line preceding the current line. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.

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

8. If an address begins with + or −, the addition or subtraction is taken with respect to the current

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

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

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

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

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

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n** or **p**, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)a

<text>

. The append command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c

<text>

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

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

e *file* The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no filename is given, the currently-remembered filename, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default filename in subsequent e, r, and w commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is not remembered as the current filename. See also *DIAGNOSTICS* below.

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

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

(1,$)g/RE/command list

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

(1,$)G/RE/ In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line character acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

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

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

(.)i

<text>

. The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)kx The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address *'x* then addresses this line; . is unchanged.

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

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

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

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

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

q The *quit* command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q The editor exits without checking for changes made in the buffer since the last *w* command.

($)r *file* The *read* command reads in the given file after the addressed line. If no filename is given, the currently-remembered filename, if any, is used (see *e* and *f* commands). The currently-remembered filename is not changed unless *file* is the very first filename mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, **$r !ls** appends current directory to the end of the file being edited. Such a shell command is not remembered as the current filename.

(.,.)s/RE/replacement/
or

(.,.)s/RE/replacement/g

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

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

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

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

copy.

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

(1,$)v/RE/command *list*

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

(1,$)V/RE/ This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file* The write command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered filename is *not* changed unless *file* is the very first filename mentioned since *ed* was invoked. If no filename is given, the currently-remembered filename, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is not remembered as the current filename.

($)= The line number of the addressed line is typed; . is unchanged by this command.

!shell *command*

The remainder of the line after the ! is sent to the system shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered filename; if a ! appears as the first character of the shell command, it is replaced with the text of the previous

shell command. Thus, **!!** repeats the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(.+1)<newline>An address alone on a line causes the addressed line to be printed. A new line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to the command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per filename, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., **/**) would be the last character before a new line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2   s/s1/s2/p
g/s1      g/s1/p
?s1       ?s1?
```

FILES

/tmp/e# temporary; *#* is the process number.

ed.hup work is saved here if the terminal is hung up.

SEE ALSO

grep(1), sed(1), sh(1), stty(1), fspec(4), regexp(5).

DIAGNOSTICS

? command errors

?*file* inaccessible file (use the *help* and *Help* commands for detailed explanations).

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

CAUTIONS AND BUGS

A ! command cannot be subject to a *g* or a *v* command.

The ! command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).

The sequence \n in a RE does not match a new-line character.

The *l* command mishandles DEL.

Characters are masked to 7 bits on input.

If the editor input is coming from a command file (i.e., **ed** *file* < *ed-cmd-file*), the editor will exit at the first failure of a command that is in the command file.

If the line disconnects while in *ed*, mail is no longer sent to the user about the creation of the file named *ed.hup*. The file itself will still be created as in past releases, either in the current directory if file and directory permissions allow, or else in the user's $HOME directory (if allowed by file and directory permissions).

NAME

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

SYNOPSIS

edit [**-r**] *file* ...

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command oriented editor. The following brief introduction should help you get started with *edit*. A more complete basic introduction is provided by tutorial materials in the *Advanced Utilities User Guide*. See *ex*(1) for other useful documents; in particular, if you are using a CRT terminal you may want to learn about the display editor *vi*.

BRIEF INTRODUCTION

To edit the contents of an existing file, begin with the following command to the shell:

edit *filename*

Edit makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but don't worry.

Edit prompts for commands with the character :, which you should see after starting the editor. If you are editing an existing file, then you have some lines in the *edit* buffer (its name for the copy of the file you are editing). Most commands to *edit* use the *current line* if you don't specify which line to use. Thus, if you type **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands), the current line is printed.

If you **delete** (d) the current line, *edit* prints 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* reads lines from your terminal, placing these lines after the current line. You terminate the input process by typing a line consisting of just a period or full stop (.). The last line of text before the . line 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, the first line having number 1. If you give the command **1** then *edit* types deletes the first line, line 2 becomes line 1, and *edit* prints the current line (the new line 1) so you can see where you are. In general, the current line is always 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 type **s**/*old*/*new*/, where *old* is replaced by the characters you want to eliminate and *new* is the new characters you want to insert.

The command **file** (f) tells you how many lines are in the buffer you are editing and says **[Modified]** if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (w) command. You can then leave the editor by issuing a **quit** (q) command. If you run *edit* on a file, but don't change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you are warned that there has been **No write since last change** and *edit* awaits 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 changes the current line to a sequence of lines you supply (as with **append** you terminate **change** with 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 reverses the effect of the last command that changed the buffer. Thus, if a **substitute** command doesn't do what you want, you can type **undo** and the old contents of the line are restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* issues a warning message when commands you give affect more than one line of the buffer. If the amount of change seems unreasonable, type *undo* and look to see what happened. If you decide that the change is ok, then you can type *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, hit carriage return. To look at a number of lines, hit **CTRL-d** (control key and, while it is held down, **d** key, then let up both) rather than carriage return. This shows you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text surrounding the current line by giving the command **z.**. The current line becomes 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 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 you can specify the number you want. For example, **z.12** produces 12 lines of text. This method of giving counts can be used with other commands. You can delete 5 lines starting with the current line with the command **delete 5**.

You can use line numbers to find things in a file; since the line numbers change when you insert and delete lines, this is somewhat unreliable. You can search backward and forward 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 current line. A useful feature 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 the symbolic name *dot* (fB.); 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 one.

You can find out the line number of the current line by typing **.=**. 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 type **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 typing **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* in the command shown above, you can get a pattern for copying lines. If the text you wish to move or copy is all within one file it is not necessary to use named buffers. **10,20move $**, for example, moves lines 10 through 20 to the end of the file.

SEE ALSO

ex (1), vi (1),
Advanced Utilities User Guide

BUGS

See *ex*(1).

NAME

enable, disable - enable/disable LP printers

SYNOPSIS

enable *printers*

disable [**-c**] [**-r**[*reason*]] *printers*

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, preventing them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers are reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

-c Cancel any requests that are currently printing on any of the designated printers.

-r[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a reason, then a default reason is used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env - set environment for command execution

SYNOPSIS

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

DESCRIPTION

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

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

SEE ALSO

sh(1), exec(2), profile(4), environ(5).

NAME

ex - text editor

SYNOPSIS

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

DESCRIPTION

Ex is the root of a family of editors which includes *edit*, *ex* and *vi*. *Ex* is a line oriented editor which is a superset of *ed*.

If you have a CRT terminal, you may wish to use the display based editor *vi* (see *vi*(1)), 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. *Ex* uses many more terminal capabilities than *ed* does. It uses the data base *terminfo*(4) and your terminal type (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 is the central mode of editing when using *vi*(1)). There is also an interline editing command, **open** (**o**), which works on all terminals.

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives access to windows of text. Hitting **CTRL-d** causes the editor to scroll a half-window of text, which is useful for quickly stepping through a file. 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 feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command. This makes it easy to detect when a command has affected more lines than you intended.

The editor normally prevents overwriting existing files so that you can't accidentally clobber a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the **recover** command to retrieve your work. This gets 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 edit each in turn. The **next** command can also be given a list of filenames, or a pattern (as used by the shell) to specify a new set of files to be edited. In general, filenames in the editor may be formed with full shell metasyntax. The metacharacter `%' is also available in forming filenames and is replaced by the name of the current file.

For 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 the 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 very useful option is *autoindent*, which allows the editor to automatically supply leading white space to align text. You can then use the **CTRL-d** key as a backtab and space and tab forward to align new code easily.

Miscellaneous new 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 Invoke *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 is printed.

+*command* Begin editing by executing the specified editor search or positioning *command*.

-l **LISP** mode; indents appropriately for lisp code. The **()** **{}** **[[** and **]]** commands in *vi* and *open* are modified to have meaning for *lisp* .

The *name* argument indicates files to be edited.

Ex States

Command Normal and initial state. Input prompt is a colon (:). The kill character cancels a partial command.

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

Open/visual Entered by **open** or **vi**; terminated with **Q** or ^.

Ex Command Names and Abbreviations

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**	open	**o**	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*	*/pat*	next line with *pat*
.	current line	?pat	previous line with *pat*
$	last line	*x-n*	*n* lines before linenumber *x*
+	next line	*x,y*	lines *x* through *y*
-	previous line	*'x*	marked with *x*
+n	*n* lines forward	*''*	previous context
%	1,$		

Initializing options

EXINIT	environmental variable for options
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option
set nox	disable option
set *x=val*	give value *val*
set	show changed options
set all	show all options

set *x*? show value of option *x*

Useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp		( ) { } are s-exp's
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
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

FILES

/usr/lib/ex?.?strings
/usr/lib/ex?.?recover
/usr/lib/ex?.?preserve
/usr/lib/*/*
$HOME/.exrc

./.exrc
/tmp/Exnnnnn
/tmp/Rxnnnnn
/usr/preserve

SEE ALSO

awk(1), ed(1), grep(1), vi(1), terminfo(4).
Advanced Utilities User Guide

WARNINGS AND BUGS

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

Undo never clears the buffer modified condition.

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

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

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

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

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

NAME

expr - evaluate arguments as an expression

SYNOPSIS

expr *arguments*

DESCRIPTION

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

The operators and keywords are listed below. Characters that need to be escaped are preceded by ****. The list is in order of increasing precedence, with equal precedence operators grouped within **{}** symbols.

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

expr **\&** *expr* returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

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

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

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

expr **:** *expr* The matching operator **:** compares the first argument with the second argument which must be a regular expression; regular expression syntax is the same as that of *ed*(1), except that all patterns are *anchored* (i.e., begin with **^**). Therefore, **^** is not a special character in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the **\(...\)** pattern symbols can be used to return a portion of the first argument.

EXAMPLES

a=`expr $a + 1`
adds 1 to the shell variable **a**.

'For $a equal to either */usr/abc/file* or just *file*
expr $a : '.*/\(.*\)' \| $a"
returns the last segment of a pathname (i.e., file).
Watch out for **/** alone as an argument: *expr*
will take it as the division operator (see BUGS below).

A better representation of example 2.
expr //$a : '.*/\(.*\)'
The addition of the **//** characters eliminates any
ambiguity about the division operator and simplifies
the whole expression.

expr $VAR : '.*'
returns the number of characters in **$VAR**.

SEE ALSO

ed(1), sh(1).

EXIT CODE

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

0 if the expression is neither null nor **0**

1 if the expression is null or **0**

2 for invalid expressions.

DIAGNOSTICS

syntax error for operator/operand errors

non-numeric argument
if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by its position. If **$a** is an =, the command:

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

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

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

NAME

factor - factor a number

SYNOPSIS

factor [*number*]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2e56 (about 7.2e16) it factors the number and prints its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

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

Maximum time to factor is proportional to sqrt(n) and occurs when *n* is prime or the square of a prime.

DIAGNOSTICS

Ouch input out of range or garbage input.

SEE ALSO

Advanced Utilities User Guide

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* prints the version stamp, provided it is greater than 0 (see *ld*(1)).

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

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.

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

The **-c** flag 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**.

FILES

/etc/magic

SEE ALSO

ld(1).

NAME

find - find files

SYNOPSIS

find pathname-list *expression*

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n is used as a decimal* integer where +*n* means more than *n*, -*n* means less than *n*, and *n* means exactly *n*.

-name *file* True if *file* matches the current filename. Normal shell argument syntax may be used if escaped (watch out for [, ? and *).

-perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

(flags&onum)==onum

-type *c* True if the type of the file is *c*, where c *is* **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (named pipe), or plain file.

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

-user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

-group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

-size *n*[**c**] True if the file is *n* blocks long (512 bytes per block). If *n* is followed by **c**, the size is given in characters.

-atime *n* True if the file has not been accessed in *n* days.

-mtime *n* True if the file has not been modified in *n* days.

-ctime *n* True if the file has not been changed in *n* days.

-exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current pathname.

-ok *cmd* Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

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

-cpio *device* Write the current file on *device* in *cpio*(A) format (5120 byte records).

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

-depth Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories

without write permission.

(*expression*) True if the parenthesized expression is ... (parentheses are special to the shell and ... escaped).

The primaries may be combined using the following ... order of decreasing precedence):

1. The negation of a primary (! is the unary ...

2. Concatenation of primaries (the *and* operation ... juxtaposition of two primaries).

3. Alternation of primaries (**-o** is the *or* operator).

NOTE: *Find* updates the time of access on the directories ... searches (i.e., it performs a **touch -a**).

EXAMPLES

1. To find a file, named *filename*, located somewhere in the directory system, use the command:

   ```
   find /  -name filename -print
   ```

2. To remove all files named *a.out* or **.o* that have not been accessed for a week use the command:

   ```
   find  \( -name a.out -o -name '*.o' \) -atime 7 -exec ...
   ```

   The braces are replaced by the located pathnames and constitute input to the *rm*(1) command.

3. The following example shows how to use the unary **not** operator to list files that have been accessed in the last 24 hours:

   ```
   find / -name '*' \(! -atime +1 \) -exec ls ...
   ```

In this way, the negation of **-atime +1** is the negation of *the files not accessed in more then 1 day*, that is files accessed in the last 24 hours. Note that the operator **!** is meaningful to the C Shell and must be escaped if the command is invoked while using the C Shell.

4. To remove files (with command *rm*(1M)) that have not been modified for a week, with the safeguard of user confirmation:

```
find . -name '*.c' -mtime +7 -o rm {} \;
```

FILES

```
/etc/passwd
/etc/group
```

SEE ALSO

cpio(1), sh(1), test(1), stat(2), cpio(4), fs(4).

NAME

getopt - parse command options

SYNOPSIS

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

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters (see *getopt*(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option -- is used to delimit the end of the options. If it is used explicitly, *getopt* recognizes it; otherwise, *getopt* generates it; in either case, *getopt* places it at the end of the options. The shell's positional parameters ($1 $2 ...) are reset so that each option is preceded by a - and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

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

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
     echo $USAGE
     exit 2
fi
for i in $*
do
```

```
        case $i in
        -a | -b) FLAG=$i; shift;;
        -o)      OARG=$2; shift 2;;
        --)              shift; break;;
        esac
    done
```

This code accepts any of the following as equivalent:

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

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

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

NAME

greek - select terminal filter

SYNOPSIS

greek [-T*terminal*]

DESCRIPTION

Greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE(Reg.) Teletypewriter Model 37 terminal (which is the *nroff* default terminal) for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable **$TERM** (see *environ*(5)). The following *terminals* are recognized currently:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

FILES

/usr/bin/300

/usr/bin/300s
/usr/bin/4014
/usr/bin/450

SEE ALSO

300(1), 4014(1), 450(1), eqn(1), mm(1), nroff(1), environ(5), greek(5), term(5).
Advanced Utilities User Guide

NAME

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

SYNOPSIS

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

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

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

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular expressions in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

-v All lines but those matching are printed.

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

-c Only a count of matching lines is printed.

-i Ignore upper/lowercase distinction during comparisons.

-l Only the names of files with matching lines are listed (once), separated by new lines.

-n Each line is preceded by its relative line number in the file

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

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

-e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a - (does not work with *grep*).

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

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

Fgrep searches for lines that contain one of the *strings* separated by new lines.

Egrep accepts regular expressions as in *ed*(1), except for .(and .), with the addition of:

1. A regular expression followed by + matches one or more occurrences of the regular expression.

2. A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.

3. Two regular expressions separated by | or a newline character match strings that are matched by either.

4. A regular expression may be enclosed in parentheses **()** for grouping.

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

SEE ALSO

ed(1), sed(1), sh(1).
Advanced Utilities User Guide

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*, but we don't know 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**.

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

If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

head - give first few lines

SYNOPSIS

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

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

SEE ALSO

tail(1)

NAME

help - ask for help

SYNOPSIS

help [*args*]

DESCRIPTION

Help finds information to explain a message from [illegible] or explain the use of a command. Zero or more arguments may be supplied.

The arguments may be either message numbers (which norma[illegible] in parentheses following messages) or command names, of one of the following types:

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **get** for message 6 from the *get* command).

type 2 Does not contain numerics (as a command [illegible]

type 3 Is all numeric (e.g., **212**)

The response of the program is the explanatory information related to the argument, if there is any.

When all else fails, try the command **help stuck**.

FILES

/usr/lib/help directory containing files of message text.

/usr/lib/help/helploc
file containing locations of help files not in **/usr/lib/help**.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

hostid - set or print identifier of current host system

SYNOPSIS

hostid [*identifier*]

DESCRIPTION

The *hostid* command prints the identifier of the current host in hexadecimal. This numeric value is expected to be unique across all hosts and is normally set to the host's Internet address. The super-user can set the hostid by giving a hexadecimal argument; this is usually done in the startup script **/etc/rc.local**.

SEE ALSO

gethostid(2), sethostid(2)

NAME

hostname - set or print name of current host system

SYNOPSIS

hostname [*nameofhost*]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the *login* prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script **/etc/rc.local**.

SEE ALSO

gethostname(2), sethostname(2)

NAME

hyphen - find hyphenated words

SYNOPSIS

hyphen [*files*]

DESCRIPTION

Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

EXAMPLE

The following allows the proof-reading of *nroff* hyphenation in *textfile*.

```
nroff textfile | hyphen
```

SEE ALSO

mm(1), troff(1).

BUGS

Hyphen can't cope with hyphenated *italic* (i.e., underlined) words; it often misses them completely or mangles them. *Hyphen* occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

id - print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

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

SEE ALSO

logname(1), getuid(2).

NAME

ipcrm - remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm removes one or more specified messages, semaphores, or shared memory identifiers. The identifiers are specified by the following *options*:

-q *msqid* removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

-m *shmid* removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-s *semid* removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

-Q *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

-M *shmkey* removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-S *semkey* removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

NAME

ipcs - report inter-process communication facilities status

SYNOPSIS

ipcs [*options*]

DESCRIPTION

Ipcs prints information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

-q Print information about active message queues.

-m Print information about active shared memory segments.

-s Print information about active semaphores.

If one of the options **-q**, **-m**, or **-s** is specified, only information about the indicated facility is printed. If none of the three options is specified, information about all three is printed.

-b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues; size of segments for shared memory; number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

-c Print creator's login name and group name. See below.

-o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues; number of processes attached to shared memory segments.)

-p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues; process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

-t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues; last *shmat* and last *shmdt* on shared memory; last *semop*(2) on semaphores.) See below.

-a Use all print *options*. This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.

-C *corefile*
Use the file corefile in place of **/dev/kmem**.

-N *namelist*
The argument is taken as the name of an alternate *namelist* (**/unix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do not determine which facilities are to be listed.

T (all) Type of facility:

q message queue;

m shared memory segment;

s semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags: The mode consists of 11 characters, interpreted as follows:

The first two characters are:

R if a process is waiting on a *msgrcv*;

S if a process is waiting on a *msgsnd*;

D if the associated shared memory segment has been removed. It disappears when the last process attached to the segment detaches it;

C if the associated shared memory segment is to be cleared when the first attach is executed;

- if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the

facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r if read permission is granted;

w if write permission is granted;

a if alter permission is granted;

- if the indicated permission is *not* granted.

OWNER (all) The login name of the owner of the facility entry,

GROUP (all) The group name of the group of the owner of the facility entry.

CREATOR(a,c) The login name of the creator of the facility entry.

CGROUP (a,c) The group name of the group of the creator of the facility entry.

CBYTES (a,o) The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o) The number of messages currently outstanding on the associated message queue.

QBYTES (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p) The process ID of the last process to send a message to the associated queue.

LRPID (a,p) The process ID of the last process to receive a message from the associated queue.

STIME (a,t) The time the last message was sent to the associated queue.

RTIME (a,t) The time the last message was received from the associated queue.

CTIME (a,t) The time when the associated entry was created or changed.

NATTCH (a,o) The number of processes attached to the associated shared memory segment.

SEGSZ (a,b) The size of the associated shared memory segment.

CPID (a,p) The process ID of the creator of the shared memory entry.

LPID (a,p) The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,t) The time the last attach was completed to the associated shared memory segment.

DTIME (a,t) The time the last detach was completed on the associated shared memory segment.

NSEMS (a,b) The number of semaphores in the set associated with the semaphore entry.

OTIME (a,t) The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names

/etc/group group names

SEE ALSO

msgop(2), semop(2), shmop(2).

BUGS

The report *ipcs* produces is only a close approximation of the real status, since information can be changed while the program is running.

NAME

join - relational database operator

SYNOPSIS

join [*options*] *file1 file2*

DESCRIPTION

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

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

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

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

These options are recognized:

-a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

-e *s* Replace empty output fields by string *s*.

-j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file.

-o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n is a file number and m* is a field number.

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

SEE ALSO

awk(1), comm(1), sort(1).
Advanced Utilities User Guide

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*(1), *comm*(1), *uniq*(1), and *awk*(1) are wildly incongruous.

NAME

kill - terminate a process

SYNOPSIS

kill [*-signo*] *PID* ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This normally kills processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the termination process are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

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

If a signal number preceded by - is given as the first argument, that signal is sent instead of terminate (see *signal*(2)). In particular, the command **kill -9 ...** is a sure kill.

SEE ALSO

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

NAME

login - sign on

SYNOPSIS

login [*name* [*env-var* ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. It is invoked by the system when a previous user has terminated the initial shell by typing a <CTRL-D> to indicate an end-of-file.

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it does not appear on the written record of the session.

At some installations, an option may be invoked that requires you to enter a second password. This occurs only for dial-up connections, and is prompted by the message **dialup password:**. Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day, if any, is printed. The user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) are initialized, and the file **.profile** in the working directory is executed, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is **-** followed by the last component of the interpreter's pathname (i.e., **-sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=your-login-directory
PATH=:/bin:/usr/bin
SHELL=last-field-of-passwd-entry
MAIL=/usr/mail/your-login-name
TZ=timezone-specification

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

L*n*=*xxx*

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people from logging into restricted shell environments and from spawning secondary shells which aren't restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
.profile	user's login profile

SEE ALSO

mail(1), newgrp(1), sh(1), su(1), passwd(4), profile(4), environ(5).

DIAGNOSTICS

Login incorrect The user name or password cannot be matched.

No shell, cannot open password file, or no directory
Consult a system programming counselor.

No utmp entry. You must exec login from the lowest level sh
You attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname - get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable **LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1).

NAME

lp, cancel - send/cancel requests to an LP line printer

SYNOPSIS

lp [**-c**] [**-d***dest*] [**-m**] [**-n***number*] [**-o***option*] [**-s**] [**-t***title*] [**-w**] *files*

cancel [*ids*] [*printers*]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no filenames are specified, the standard input is assumed. The filename - stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* are specified is the same order in which they are printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with filenames:

-c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* are not copied, but are linked whenever possible. If the **-c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **-c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

-d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request is printed only on that specific printer. If *dest* is a class of printers, then the request is printed on the first available printer that is a member of the class. Under certain conditions (e.g., printer unavailability, file space limitation), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set); otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

-m Send mail (see *mail*(*1*)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

-n*number* Print *number* copies (default of 1) of the output.

-o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **-o** keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin*(1M).

-s Suppress messages from *lp*(1) such as **request id is ....**

-t*title* Print *title* on the banner page of the output.

-w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail is sent instead.

Cancel cancels line printer requests made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request that is currently printing on that printer. In either case, the

cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1). accept(1M), lpadmin(1M), lpsched(1M) in the *System Administration Utilities Reference Manual*, *Advanced Utilities Reference Manual*.

NAME

lpstat - print LP status information

SYNOPSIS

lpstat [*options*]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *id*'s (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

-u "user1, user2, user3"

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed. For example,

lpstat -o

prints the status of all output requests.

-a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

-c[*list*] Print class names and their members. *List* is a list of class names.

-d Print the system default destination for *lp*.

-o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *id*'s.

-p[*list*] Print the status of printers. *List* is a list of printer names.

-r Print the status of the LP request scheduler

-s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

-t Print all status information.

-u[*list*] Print status of output requests for users. *List* is a list of login names.

-v[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls - list contents of directories

SYNOPSIS

ls [**-RadCxmlnogrtucpFbqisf**] *names*

DESCRIPTION

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

There are three major listing formats. The default format is to list one entry per line; the **-C** and **-x** options enable multi-column formats; and the **-m** option enables stream output format in which files are listed across the page, separated by commas. To determine output formats for the **-C**, **-x**, and **-m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

The following options can be used with *ls*:

-R Recursively list subdirectories encountered.

-a List all entries; usually entries whose names begin with a period (.) are not listed.

-d If an argument is a directory, list only its name (not its contents); often used with **-l** to get the status of a directory.

-C Multi-column output with entries sorted down the columns.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format.

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

-n The same as **-l**, except that the owner's *UID* and group's *GID* numbers are printed, rather than the associated character strings.

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

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

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

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

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

-c Use time of last modification of the inode (e.g., file created, mode changed) for sorting (**-t**) or printing (**-l**).

-p Put a slash (/) after each filename if that file is a directory.

-F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.

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

-q Force printing of non-graphic characters in filenames as the character (?).

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

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

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

The mode printed under the **-l** option consists of 10 characters that are interpreted as follows:

The first character is:

d if the entry is a directory

b if the entry is a block special file

c if the entry is a character special file

p if the entry is a fifo (a.k.a. "named pipe") special file

- if the entry is an ordinary file

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to

permissions of others in the user group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, execute permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r if the file is readable

w if the file is writable

x if the file is executable

- if the indicated permission is *not* granted

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **-**) is **t** if the 1000 (octal) bit of the mode is on; refer to *chmod*(1) for the meaning of this mode. The indications of set-ID and 1000 bit of the mode are capitalized (**S** and **T**, respectively) if the corresponding execute permission is *not* set.

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

FILES

/etc/passwd	user IDs for **ls -l** and **ls -o**.
/etc/group	group IDs for **ls -l** and **ls -g**.
/usr/lib/terminfo/*	terminal information.

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in filenames may confuse the columnar output options.

NAME

m4 - macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a filename is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

-e Operate interactively. Interrupts are ignored and the output is unbuffered.

-s Enable line sync output for the C preprocessor (#line ...)

-B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

-H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

-S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non- macro arguments take one.

-T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any filenames and before any **-D** or **-U** flags:

-D*name*[=*val*]

Defines *name* to *val* or to null in the absence of *val*.

-U*name* undefines *name*.

Macro calls have the form:

name(*arg1*,*arg2*, ..., *argn*)

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is assumed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit. *M4* ignores leading unquoted blanks, tabs, and new-line characters while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses that appear within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define — the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **$***n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the

arguments separated by commas; **$@** is like **$***, but each argument is quoted (with the current quotes).

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

defn returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-in ones.

pushdef like *define*, but saves any previous definition.

popdef removes current definition of its argument(s), exposing the previous one if any.

ifdef if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on the X/OS System versions of *m4*.

shift returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that is subsequently performed.

changequote changes quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., `').

changecom changes left and right comment markers from the default # and new line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

divert *m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current

stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert causes immediate output of text from diversions named as arguments, or from all diversions if no argument is present. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum returns the value of the current output stream.

dnl reads and discards characters up to and including the next new-line character.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr returns the value of its argument decremented by 1.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr — returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit — transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include — returns the contents of the file named in the argument.

sinclude — is identical to *include*, except that it says nothing if the file is inaccessible.

syscmd — executes the system command given in the first argument. No value is returned.

sysval — is the return code from the last call to *syscmd*.

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

m4exit — causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.

m4wrap — pushes back argument 1 at final EOF; example:

```
m4wrap(`cleanup()')
```

errprint — prints its argument on the diagnostic output file.

dumpdef — prints current names and definitions, for the named items, or for all if no arguments are given.

traceon — with no arguments, turns on tracing for all macros (including built-in ones); otherwise, it turns on tracing for named macros.

traceoff — turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced

only by specific calls to *traceoff*.

SEE ALSO

cc(1), cpp(1).

Common Development Tools User Manual

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

NAME

pdp11, u3b, vax, m68k - provide truth value about your processor type

SYNOPSIS

pdp11

u3b

vax

m68k

DESCRIPTION

The following commands return a true value (exit code of 0) if you are on the processor that the command name indicates.

pdp11 True if you are on a PDP-11/45 or PDP-11/70.

u3b True if you are on a 3B20S.

vax True if you are on a VAX-11/750 or VAX-11/780.

m68k True if you are on a Motorola M68000 processor.

The commands that do not apply return a false (non-zero) value. These commands are often used within *make*(1) makefiles and shell procedures to increase portability.

SEE ALSO

sh(1), test(1), true(1).

NAME

mail, rmail - send mail to users or read mail

SYNOPSIS

mail [**-epqr**] [-f file]
mail [**-t**] persons
rmail [**-t**] persons

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input to determine the disposition of the message:

newline	Go on to next message.
+	Same as **newline**.
d	Delete message and go on to next message.
p	Print message again.
-	Go back to previous message.
s [*files*]	Save message in the named *files* (**mbox** is default).
w [*files*]	Save message, without its header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons* (yourself is default).

q Put undeleted mail back in the *mailfile* and stop.

EOT (CTRL-d) Same as **q**.

x Put all mail back in the *mailfile* unchanged and stop.

!*command* Escape to the shell to do *command*.

***** Print a command summary.

The optional arguments alter the printing of the mail:

-e causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

-p causes all mail to be printed without prompting for disposition.

-q causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.

-r causes messages to be printed in first-in, first-out order.

-f*file* causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a >. The **-t** option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and

resending.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** interprets that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file is preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which causes all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the *mailfile*

$HOME/mbox	saved mail
$MAIL	variable containing pathname of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.*lock*	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), uucp(1C), write(1).

BUGS

Race conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

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.

Incoming mail is stored in a standard file for each user, called the system *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 (refer to "MBOX" description under the heading ENVIRONMENT VARIABLES below). Messages remain in this file until forcibly removed.

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 *mailbox*. Command line options are:

-d Turn on debugging output. This option is not recommended; the output is not useful.

-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 *mailbox*. If no *filename* is specified, the *mbox* is used.

-F Record the message in a file named after the first recipient. Overrides the "record" variable, if set (refer to ENVIRONMENT VARIABLES below).

-h *number* The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.

-H Print header summary only.

-i Ignore interrupts. Refer to "ignore" in the description of ENVIRONMENT VARIBLES below.

-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.

-s *subject* Set the Subject header field to *subject*.

-u *user* Read *user's mailbox*. This is only effective if *user's mailbox* is not read protected.

-U Convert *uucp* style addresses to internet standards. Overrides the **conv** enviroment variable.

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 (refer to 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. 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. Refer to TILDE ESCAPES below 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 that are set and cleared via the **set** and **unset** commands. Refer to ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If 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 (refer to 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, the **print** command 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 > in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications, 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. Filenames, where expected, are expanded via the normal shell conventions (refer to *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. Most regular commands are legal inside start-up files,

the most common use being 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**. Any errors in the start-up file cause the remaining lines in the file to be ignored.

COMMANDS

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

!shell-*command* Escape to the shell. (Refer to "SHELL" in the section below that describes ENVIRONMENT VARIABLES.).

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

= Print the current message number.

? Print a summary of commands.

alias *alias name ...*
and

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

alternates *name ...*
Declare a list of alternate names for a user's 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. Refer also to *allnet* in the section describing ENVIRONMENT VARIABLES.

cd [*directory*] and

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

copy [*filename*]
and

copy [*msglist*] *filename*
Copy messages to the file without marking the messages as saved (otherwise equivalent to the **Save** command).

Copy [*msglist*] Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved (otherwise equivalent to the **Save** command).

delete [*msglist*]
Delete messages from the *mailbox*. If **autoprint** is set, the next message after the last one deleted is printed (refer to ENVIRONMENT VARIABLES below).

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

ignore [*header-field* ...]
Suppress printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are *status* and *cc*. The fields are included when the message is saved. The **Print** and **Type** commands override this command.

dp [*msglist*] and

dt [*msglist*] Delete the specified messages from the *mailbox* and print the next message after the last one deleted (roughly equivalent to a **delete** command followed by a **print** command).

echo *string* ...
Echo the given strings (like *echo*(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 (refer to ENVIRONMENT VARIABLES). The default editor is *ed*(1).

exit and

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

file [*filename*]
and

folder [*filename*]
Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as filenames, with the following substitutions:

%	the current *mailbox*
%*user*	the *mailbox* for *user*
#	the previous file
&	the current *mbox*

Default file is the current mailbox.

folders Print the names of the files in the directory set by the **folder** variable (refer to ENVIRONMENT 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 Followup, Save, and Copy commands and the **outfolder** environment variable.

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 **followup**, **Save**, and **Copy** commands and the **outfolder** environment variable.

from [*msglist*] Print the header summary for the specified messages.

group *alias name ...*
and

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*]
Print the page of headers that includes the message specified. The **screen** variable sets the number of headers per page (refer to ENVIRONMENT VARIABLES below). See also the **z** command.

help Print a summary of commands.

hold [*msglist*] and

preserve [*msglist*]
Hold the specified messages in the *mailbox*.

if s|r

mail-commands

else

endif Conditional execution, where *s* will execute the *mail-commands* that follow, 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 ...*
and

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

list Print all commands available. No explanation is given.

mail *name ...* Mail a message to the specified users.

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

next [*message*] Go to the 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*]
and

|[*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** environment variable. If the **page** variable is set, a form-feed character is inserted after each message (refer to the section describing ENVIRONMENT VARIABLES).

preserve [*msglist*]
and

hold [*msglist*] Preserve the specified messages in the *mailbox*.

Print [*msglist*] and

Type [*msglist*] Print the specified messages on the screen, including all header fields (overrides suppression of fields by the **ignore** command).

print [*msglist*] and

ttype [*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) (refer to ENVIRONMENT VARIABLES).

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

Reply [*msglist*] and

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 filename, the

response is saved at the end of that file (refer to ENVIRONMENT VARIABLES).

reply [*message*] and

respond [*message*]
Reply to the specified message, including all other recipients of the message. If **record** is set to a filename, the response is saved at the end of that file (refer to ENVIRONMENT VARIABLES).

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

save [*filename*] and

save [*msglist*] *filename*
Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless the **keepsave** variable is set (refer to ENVIRONMENT VARIABLES and the **exit** and **quit** commands).

set **and**

set *name* *and*

set *name*=string
and

set *name*=number
Define a variable called *name*. The variable may be given a null, string, or numeric value. **Set** by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed

descriptions of the *mailx* variables.

shell Invoke an interactive shell (refer to **SHELL** in the description of ENVIRONMENT VARIABLES).

size [*msglist*] Print the size in characters of the specified messages.

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

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

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

Type [*msglist*] and

Print [*msglist*]
Print the specified messages on the screen, including all header fields (overrides suppression of fields by the **ignore** command).

ttype [*msglist*]
and

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) (refer to the section describing ENVIRONMENT 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 (refer to ENVIRONMENT VARIABLES).

unset *name* ... Erase the variables specified by *name*. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version Print 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 (refer to ENVIRONMENT VARIABLES).

write [msglist] *filename*
Write the given messages on the specified file, minus the header and trailing blank line (otherwise equivalent to the **save** command).

xit and

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

z[**+**|**-**] Scroll the header display forward or backward one screen. The number of headers displayed is set by the **screen** variable (refer to ENVIRONMENT VARIABLES).

TILDE ESCAPES

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

character.

~! *shell-command*
Escape to the shell.

~. Simulate end-of-file (terminate message input).

~: *mail-command*
and

~_*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 (refer to ENVIRONMENT VARIABLES).

~**a** Insert the autograph string **sign** into the message (refer to ENVIRONMENT VARIABLES).

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

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

~**d** Read in the **dead.letter** file. Refer to the **DEAD** variable (ENVIRONMENT VARIABLES) for a description of this file.

~**e** Invoke the editor on the partial message. Refer to the **EDITOR** variable (ENVIRONMENT VARIABLES).

~**f** [*msglist*] Forward the specified messages. The messages are inserted into the message without alteration.

~**h** Prompt for Subject line and *to*, *cc*, and *bcc* lists. If the field is displayed with an initial value, it may be edited as if it had just been typed.

~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**. Refer to the **DEAD** variable (ENVIRONMENT VARIABLES) for a description of this file.

~r *filename* and

~< *filename* and

~< !*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. Refer also to the **VISUAL** variable (ENVIRONMENT VARIABLES).

~w *filename* Write the partial message onto the given file, without the header.

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

~| *shell-command*

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

ENVIRONMENT VARIABLES

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

allnet All network names whose last component (login name) matches are treated as identical. This causes the *msglist* message specifications to behave similarly. default is **noallnet**. Refer also to 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*(1) 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. Refer also to **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 or delivery errors. 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** of **~e** command is used. Default is *ed*(1).

escape=c Substitute *c* for the **~** escape character.

folder=*directory*

The directory for saving standard mail files. User-specified filenames beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. 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*(1) environment variable. There is no default for the **folder** variable. Refer also to **outfolder** below.

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

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

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

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

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

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

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

metoo If a user's login appears as a recipient of a message, it is not deleted 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 Cause the files used to record outgoing messages to be located in the directory specified by the **folder** variable unless the pathname is absolute. Default is **nooutfolder**. Refer to the **folder** variable above and the **S**ave, **C**opy, **f**ollowup, and **F**ollowup commands.

page Used with the **pipe** command to insert a form-feed character 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. Refer also to the **outfolder** variable above.

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

screen=*number* Set the number of lines in a full screen 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 the message is from the user displaying the header summary, the recipient's name is printed 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 (refer also to **~i** (TILDE ESCAPES)).

Sign=*string* The variable inserted into the text of a message when the **~A** command is given. No default (refer also to **~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

$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	global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

mail(1), pg(1), ls(1).
Advanced Utilities User Guide

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 period (.) are treated as the end of the message by *mail*(1) (the standard mail delivery program).

In the current release, the **-r**, **-h**, and **-U** options are not

implemented.

The **echo** command provided by *mailx* does not recognize shell parameters (e.g., $$, $*, $#).

Errors in the user's **.mailrc** file are not handled correctly. The information provided should be treated as suspicious.

The tilde character, used for various escape options, cannot be changed during an escape while writing mail.

It should be noted that the subject line is limited to 1024 characters.

Mail read from the normal mailbox, **/usr/mail/$LOGNAME**, will be deleted when saved in another file (e.g., **mbox**); however, mail read from alternate mailboxes such as **$HOME/mbox** and saved elsewhere will not be deleted.

The *mailx* utility cannot be used on a remote system if **net** was used to get to that system.

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 uppercase and lowercase 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, the input and output are pipes.

SEE ALSO

passwd(4).

NAME

mesg - permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir - make a directory

SYNOPSIS

mkdir *dirname* ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

sh(1), rm(1), umask(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

NAME

mklib - build a shared library file

SYNOPSIS

mklib [*assembly-source* ...]

DESCRIPTION

The tool called *mklib* is used to convert assembly language source files generated by the C compiler (through the use of the **-S** compile-time option) into an assembly source that is self-relocating (position independent) and that places all of the statically allocated data into the *text* section.

The output file is always the standard output. Therefore, in normal usage, the output will have to be redirected to the appropriate file. In case no input files are listed, the tool takes the standard input as its input. Otherwise, the list of *assembly-source* files is taken as input.

Note that whenever C sources are compiled into assembly sources using the C compiler, the *-X39* compilation switch should be set, to avoid optimizations that may prevent *mklib* from working properly.

During normal processing, the tool transforms all references to absolute labels that appear within the source files into program counter-relative references, so that the output code is effectively position-independent. This transformation may require the replacement of the *jmp* instruction with a *bra*, that of *jsr* instructions with a *bsr* (unless the *jmp* or the *jsr* use the register indirect mode), and so on.

The outcome of the instruction replacement is meaningful only in case the assembler sees the symbol as appearing in the same file where it is referenced. This implies that such substitutions can be

carried out for all the symbols that exist in any of the input files. Such symbols will eventually appear in the output of the tool (that will be used as input to the assembler) and for such reason can be correctly handled by the assembler.

Symbols not defined within one of the input files cannot be addressed as described. Symbols in this class will be defined by the linker and references to them need not be program counter-relative.

Moreover, the program takes care of moving the statically allocated data to the text section by removing all the "data" assembler pseudo-operations, collecting the data declarations, along with the associated labels and placing them outside of the code for each procedure. This will avoid the generation of additional jump instructions within procedures, for the purpose of skipping the data. All the data items relevant to a procedure are placed *before* the procedure itself. Note that, besides solving other problems, this preserves locality of references (an important performance issue in demand-paged systems). Note that care is taken to align all the data structures to longword boundaries.

In case multiple source files are supplied in input, the tool makes sure that the relative ordering among procedures and that among procedures and data areas stays unchanged. For example, in case the first object in the first input file is a procedure, this procedure will be output before any other item.

Note that, where necessary, the tool takes care of changing all the internal labels and symbols to prevent conflicts across the various input assembly files. One more step taken in the case of multiple input files is that of removing all of the "file" directives, except for the first one.

SEE ALSO

LSX X/OS Programmer Guide

NAME

ncpio - copy file archives in and out

SYNOPSIS

/usr/public/ncpio -o [**ac{B|F|T}vLMVA***nnn*] <
name-list > *collection*

/usr/public/ncpio -i [**{B|F|T}cdmrtuvfsSb6MA***nnn*]
[*patterns*] < *collection*

/usr/public/ncpio -p [**adlmruvL**] *directory* < *name-list*

DESCRIPTION

Ncpio -o (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information.

Ncpio -i (copy in) extracts files from the standard input which is assumed to be the product of a previous **ncpio -o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1). In *patterns*, metacharacters **?**, *****, and **[...]** match the slash **/** character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is ***** (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

Ncpio -p (pass) reads the standard input to obtain a list of pathnames of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

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

B Block input/output 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from **/dev/rmt?**).

d Create *directories* as needed.

c Write *header* information in ASCII character form for portability.

r Interactively *rename* files. If the user types a null line, the file is skipped.

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

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

v *Verbose*: print a list of filenames. When used with the **t** option, the table of contents looks like the output of an **ls -l** command (see *ls*(1)).

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

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

f Copy in all files except those in *patterns*.

s Swap bytes. Use only with the **-i** option.

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

b Swap both bytes and halfwords. Use only with the **-i** option.

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

The following options invalidate the **cpio -oc** portability on other systems.

A Skip *nnn* bytes.

F I/O using **/dev/rfd[01]b** (multifloppy). The size of each I/O block matches that of 1 cylinder on floppy.

M Mark a multifloppy sequence (must follow **-F**). Floppies written with **-M**, can only be read with **-M**.

V Verify (must follow **-o** and **-F**).

T I/O using **/dev/rmt8** or **/dev/rstc5**. The I/O block-size is 16Kbytes.

L Recognize symbolic links.

EXAMPLES

To save **dir** on floppy:

```
cd dir; find . -depth -print | /usr/public/ncpio -oacvLMF > dev/rfd0b
```

To restore in **ndir**:

```
cd ndir; /usr/public/ncpio -icdmvMF < /dev/rfd0b
```

SEE ALSO

cpio(1), cpio(4).

BUGS

Pathnames are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory; thereafter, linking information is lost. Only the superuser can copy special files.

NAME

newform - change the format of a text file

SYNOPSIS

newform [**-s**] [-itabspec] [-otabspec] [**-bn**] [**-en**] [**-pn**] [**-an**] [**-f**] [**-c***char*] [**-ln**] [*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 **-e15 -l60** yield results different from **-l60 -e15**. Options are applied to all *files* on the command line.

-itabspec Input tab specification: expands tabs to spaces, according to the tab specifications given.

Tabspec recognizes all tab specification forms described in *tabs*(1). If *tabspec* is --, *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**.

-otabspec Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for fB-itabspec. If no *tabspec* is given, *tabspec defaults to* **-8**. A *tabspec* of

-0 means that no spaces will be converted to tabs on output.

-l*n* 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).

-b*n* Truncates *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-l***n*). 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 filename
```

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.

-e*n* Same as **-b***n* except that characters are truncated from the end of the line.

-c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

-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 printed tab specification format line corresponds to the format

specified in the last **-o** option. If no **-o** option is specified, the printed line contains the default specification of **-8**.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit 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, use the command:

newform -s -i -1 -a -e *filename*

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).

EXIT CODES

0 - normal execution

1 - for any error

SEE ALSO

csplit(1), tabs(1), fspec(4).
Advanced Utilities User Guide

BUGS

Newform normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* keeps track of backspaces in order to line up tabs in the appropriate logical columns.

Newform does not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was **-o--**, and was preceded by either **-o--** or **-i--**, the tab specification format line will be incorrect.

NAME

newgrp - log in to a new group

SYNOPSIS

newgrp [-] [group]

DESCRIPTION

Newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell by *newgrp* (replacing the current shell), regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (**PS1**) other than **$** (default) and has not exported **PS1**. After an invocation of *newgrp*, successful or not, the **PS1** will now be set to the default prompt string, **$**. Note that the shell command *export* (refer to *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

If the first argument to *newgrp* is a -, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user

does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

news - print news items

SYNOPSIS

news [**-a**] [**-n**] [**-s**] [items]

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. Afterwards, *news* stores the "current" 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 are considered "current" by the next invocation of *news*.

The **-a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The **-n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The **-s** option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile

/usr/news/*

$HOME/.news_time

SEE ALSO

profile(4), environ(5).
Advanced Utilities User Guide

NAME

nice - run a command at low priority

SYNOPSIS

nice [-increment] command [arguments]

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The superuser may run commands with priority higher than normal by using a negative increment, e.g., **--10.**

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl - line numbering filter

SYNOPSIS

nl [**-h**type] [**-b**type] [**-f**type][**-v**start#] [**-i**incr] [**-p**] [**-l**num] [**-s**sep] [**-w**width] [**-n**format] [**-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 filename. Only one file may be named. The options are:

-btype Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p**string, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is Bt (text lines numbered).

-htype Same as **-b**type except for header. Default *type* for logical page header is **n** (no lines numbered).

-ftype Same as **-b**type except for footer. Default for logical page footer is **n** (no lines numbered).

-p Do not restart numbering at logical page delimiters.

-vstart# *Start#* is the initial value used to number logical page lines. Default is **1**.

-iincr *Incr* is the increment value used to number logical page lines. Default is **1**.

-ssep *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

-wwidth *Width* is the number of characters to be used for the line number. Default *width* is **6**.

-nformat *Format* is the line numbering format. Recognized values are: **ln**, left-justified, leading zeroes suppressed; **rn**, right-justified, leading zeroes suppressed; **rz**, right-justified, leading zeroes kept. Default *format* is **rn** (right-justified).

-lnum *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**.

-dxx 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
```

numbers file 1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nohup - run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* with hangups and quits ignored. If output is not redirected by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

It is frequently necessary to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. If the following command is given:

```
nohup sh file
```

the **nohup** applies to everything in **file**. If the shell procedure **file** is to be executed often, then the need to type **sh** can be eliminated by giving **file** execute permission. Add an ampersand and the contents of **file** are run in the background with interrupts also ignored (refer to *sh*(1)):

```
nohup file &
```

The contents of **file** could be:

```
tbl ofile | eqn | nroff > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

CAUTION

In the command line

nohup command1; command2

the *nohup* applies only to *command1*. The command

nohup (command1; command2)

is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape:

nohup cpio -o < list > /dev/rmt/1m&

The command

nohup cpio -o < list > /dev/rmt/1m 2>errors&

puts the error messages into a file named **errors**.

NAME

od - octal dump

SYNOPSIS

od [**-bcdosx**] [file] [[**+**]offset[**.**][**b**]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **-o** is default. The meanings of the format options are:

-b Interpret bytes in octal.

-c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**\0**, backspace=**\b**, form feed=**\f**, new line=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.

-d Interpret words in unsigned decimal.

-o Interpret words in octal.

-s Interpret 16-bit words in signed decimal.

-x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If

the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.

An asterisk (*) appearing on a line by itself in the output indicates that the previous line is repeated until the next full line. This output commonly occurs when *od* is used on a file that contains multiple nulls.

SEE ALSO

dump(1).
Advanced Utilities User Guide

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* is 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* 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 occurs if:

1. the file appears to be already packed;

2. the filename has more than 12 characters;

3. the file has links;

4. the file is a directory;

5. the file cannot be opened;

6. no disk storage blocks will be saved by packing;

7. a file called *name*.z already exists;

8. the .z file cannot be created; or

9. an I/O error occurred during processing.

The last segment of the filename must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.z use:

pcat name.z

or just:

pcat name

To make an unpacked copy, *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:

1. the filename (exclusive of the .z) has more than 12 characters;

2. the file cannot be opened; or

3. the file does not appear to be the output of *pack*.

Unpack expands files created by *pack*. For each *filename* 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:

1. a file with the "unpacked" name already exists; or

2. the unpacked file cannot be created.

SEE ALSO

cat(1)

NAME

passwd - change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the login *name*.

Ordinary users may change only the password that corresponds to their login *name*.

The program prompts for the old password (if any) and then for the new one (twice). The first time the new password is entered, *passwd* checks to see if the old password has "aged" sufficiently. If "aging" is insufficient, the new password is rejected and *passwd* terminates; refer to *passwd*(4).

Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical, the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

1. Each password must have at least six characters. Only the first eight characters are significant.

2. Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means uppercase and lowercase letters.

3. Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

4. New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

A user whose effective user ID is zero is called a superuser; refer to *id*(1) and *su*(1). Superusers may change any password; hence, *passwd* does not prompt superusers for the old password. Superusers are not forced to comply with password aging and password construction requirements. A superuser can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

id(1), login(1), su(1), crypt(3C), passwd(4).

NAME

paste - merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...

paste -dlist 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). *Paste* is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if - is used in place of a filename.

The meanings of the options are:

-d Replace the *tab* character by one or more alternate characters specified in *list*. 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.

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 the **-d** option. Regardless of the *list*, the last character of the file is forced to be a new line.

- May be used in place of any filename, to read a line from the standard input. (There is no prompting).

EXAMPLES

```
ls | paste -d "" -
```

lists directory in one column

```
ls | paste - - - -
```

lists directory in four columns

```
paste -s -d "\t\n" file
```

combines pairs of lines into lines

SEE ALSO

grep(1), cut(1),

pr(1): **pr -t -m...** works similarly, but creates extra blanks, tabs and new lines for a nice page layout.

DIAGNOSTICS

line too long Output lines are restricted to 511 characters.

too many files Except for the **-s** option, no more than 12 input files may be specified.

NAME

pg - file perusal filter for soft-copy terminals

SYNOPSIS

pg [-number] [**-p** string] [**-cefns**] [+linenumber] [+/pattern/] [files...]

DESCRIPTION

The *pg* command is a filter that allows the examination of *files* one screen at a time on a soft-copy terminal. The filename - and/or NULL arguments indicate that *pg* should read from the standard input. Each full screen 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 because it allows the user to back up and review something that has already passed. The method for doing this is explained below.

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 new-line 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 that 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 — These cause 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 full screen beginning at the specified line.

(+1) **d** or

CTRL-d — Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **CTRL-l** — 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 character, even if the **-n** option is specified.

i/pattern/ — Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wraparound.

i^pattern^ and

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 wraparound. The ^ notation is useful for Adds 100 terminals that 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:

in Begin perusing the *i*th next file in the command line. The *i* is an unsigned number; default value is 1.

ip Begin perusing the *i*th previous file in the command line. The *i* is an unsigned number; default is 1.

iw 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 new-line character, 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 new- line character, even if the -n option is specified.

At any time when output is being sent to the termianl, the user can hit the quit key (normally **CTRL-**) or the interrupt (**BREAK**) key. This causes *pg* to stop sending output and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, 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* command 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 data base
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1), terminfo(4).

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

pr - print files

SYNOPSIS

pr [options] [files]

DESCRIPTION

Pr prints the named files on the standard output. If *file* is -, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

The *options* below may appear singly or be combined in any order:

+*k* Begin printing with page *k* (default is 1).

-*k* Produce *k*-column output (default is 1). The options **-e** and **-i** are assumed for multi-column output.

-a Print multi-column output across the page.

-m Merge and print all files simultaneously, one per column (overrides the **-k**, and **-a** options).

-d Double-space the output.

-eck Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

-ick In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

-nck Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **-m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

-wk Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).

-ok Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

-lk Set the length of a page to *k* lines (default is 66).

-h Use the next argument as the header to be printed instead of the filename.

-p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

-f Use form-feed character for new pages (default is to use a sequence of line feeds). Pause before beginning the first page if the standard output is associated with a terminal.

-r Print no diagnostic reports on failure to open files.

-t Print neither the 5-line identifying header nor the 5-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

-s*c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr -e9 -t < file1 > file2
```

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1).

NAME

printenv - print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

If a *name* is specified and it is not defined in the environment, *printenv* returns exit status 1, else it returns status 0.

SEE ALSO

sh(1), environ(5), csh(1)

NAME

ps - report process status

SYNOPSIS

ps [options]

DESCRIPTION

Ps prints information about active processes. Without *options*, information is printed about processes associated with the current terminal. Otherwise, the displayed information is controlled by the following *options*:

-e Print information about all processes.

-d Print information about all processes, except process group leaders.

-a Print information about all processes, except process group leaders and processes not associated with a terminal.

-f Generate a *full* listing. Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed. See below for meaning of columns in a full listing.

-l Generate a *long* listing. See below.

-c *corefile* Use the file *corefile* in place of **/dev/mem**.

-s *swapdev* Use the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*; a *swapdev* of **/dev/null** causes the user block to be zeroed out.

-n *namelist* The argument is taken as the name of an alternate *namelist* (**/unix** is the default).

-t *termlist* Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified in one of two forms: the device filename (e.g., **tty-4**) or, if the device filename starts with **tty**, just the digit identifier (e.g., **04**).

-p *proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.

-u *uidlist* Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID is printed unless the **-f** option is used, in which case the login name is printed.

-g *grplist* Restrict listing to data about processes whose process groups are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options only determine what information is provided for a process; they do *not* determine which processes are to be listed.

F (1) Flags (octal and additive) associated with the process:

01	in core
02	system process
04	locked in core (e.g., for physical I/O)
10	being swapped
20	being traced by another process
40	another tracing flag

S (1) The state of the process:

		O non-existent S sleeping W waiting R running I intermediate Z terminated T stopped X growing
UID	(f,l)	The user ID number of the process owner; the login name is printed under the **-f** option.
PID	(all)	The process ID of the process; it is possible to kill a process if you know the **PID**.
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
STIME	(f)	Starting time of the process.
PRI	(l)	The priority of the process; higher numbers mean lower priority.
NI	(l)	Nice value; used in priority computation.
ADDR	(l)	The memory address of the process (a pointer to the segment table array on the 3B20S), if resident; otherwise, the disk address.
SZ	(l)	The size in blocks of the core image of the process.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
TTY	(all)	The controlling terminal for the process.
TIME	(all)	The cumulative execution time for the process.

CMD (all) The command name; the full command name and its arguments are printed under the **-f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **-f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **-f** option, is printed in square brackets.

FILES

/unix	system namelist.
/dev/mem	memory.
/etc/passwd	supplies UID information.
/etc/ps_data	internal data structure.
/dev	searched to find terminal ("tty") names.

SEE ALSO

acctcom(1), kill(1), nice(1).

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

NAME

ptx - permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

Ptx generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. *Ptx* output is in the form:

```
.xx "tail" "before keyword" "keyword and after"
"head"
```

where **.xx** is assumed to be an *nroff* or *troff*(1) macro provided by the user, or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

-f Fold upper and lower case letters for sorting.

-t Prepare the output for the phototypesetter.

-w *n* Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for

nroff and 100 for *troff*.

-g *n* Use the next argument, *n*, as the number of characters that *ptx* reserves in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.

-o *only* Use as keywords only the words given in the *only* file.

-i *ignore* Do not use as keywords any words given in the *ignore* file. If the **-i** and **-o** options are missing, use **/usr/lib/eign** as the *ignore* file.

-b *break* Use the characters in the *break* file to separate words. Tab, new-line, and space characters are always used as break characters.

-r Take any leading non-blank characters of each input line to be a reference identifier (e.g., a page or chapter reference), separate from the text of the line. Attach the reference identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

FILES

/bin/sort

/usr/lib/eign

/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), troff(1), mm(5), mptx(5).

BUGS

Line length counts do not account for overstriking or proportional

spacing.

Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

NAME

pwd - working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

Cannot open .. and **Read error in ...** indicate possible file system trouble. These messages should be referred to a system programming counselor.

NAME

regcmp - regular expression compile

SYNOPSIS

regcmp [-] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.**i**. If the - option is used, the output is placed in *file*.**c**. The format of entries in *file* is a name (C variable), followed by one or more blanks, followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File*.**i** files may thus be *included* into C programs, or *file*.**c** files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) applies the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

The following is the regular expression, contained in a file called **BROWN**, corresponding to the string *brown*, making no distinction between upper and lower case characters:

```
name      "([Bb][Rr][oO][wW][nN][A-Za-z ]+\n)$0"
```

The command line:

```
regcmp BROWN
```

compiles the regular expression and places the output in a file called *BROWN.i*. To search for all instances of *brown*, *BROWN*, *browN*,

etc, in a file called *NAME*, the C program to be included in the output file is:

```
/* Extern and include statement */
.
.
.
#include "BROWN.i"

  main() {
     int i = 0;
     char *ADDRESS;
     int fd;
     int rd = 1;
     char BUF[512];
     char locl[20];

     fd = open("NAME", O_RDONLY);

     while (rd != 0) {
        rd = read(fd, BUF, 511);
        BUF[rd] = '\0';
        ADDRESS = regex(name, BUF, locl);
        while ( ADDRESS != 0) {
            printf("%s\n", locl);
            ADDRESS = regex (name, ADDRESS, locl);
            }
        }
        close(fd);
    }
```

The subroutine *regex*(3X) applies the regular expression *name* to *ADDRESS*.

SEE ALSO

regcmp(3X).

NAME

rm, rmdir - remove files or directories

SYNOPSIS

rm [**-fri**] file ...
rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry is the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with **y** the file is deleted; otherwise, the file remains. No questions are asked when the **-f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used. When **-r** is used, *rm* recursively deletes the entire contents of the specified directory, and the directory itself. To remove a directory, you must be located at the next higher level (parent directory) in the directory hierarchy. You cannot be located in the directory you are trying to remove. For example, if the full pathname of a directory is **/top/next**, you must be in **/top** to remove the directory **next**.

If the **-i** (interactive) option is in effect, *rm* asks whether to delete each file. The name of the file is printed on the terminal followed by a colon. If you do not want to delete the file, press the carriage return. If you do want to delete the file, type input that begins with the letter "**y**". The **-i** option can be used in combination with the **-r** option to interactively delete directories.

When the command **rm -ir** *directory name* is given, the message **directory** *directory name*: is printed on the terminal. As with interactive file deletion, type either input beginning with the letter **y** or a carriage return alone to delete or not delete a directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2).

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file .. merely to avoid the consequences of inadvertently doing something like:

```
rm -r .*
```

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 produced by *diff*(1) are printed, where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a **%** and waits for one of the following user-typed commands:

l append the left column to the output file

r append the right column to the output file

s turn on silent mode; do not print identical lines

v turn off silent mode

e l call the editor with the left column

e r call the editor with the right column

e b call the editor with the concatenation of left and right

e call the editor with a zero length file

q exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed - stream editor

SYNOPSIS

sed [**-n**] [**-e** script] [**-f** sfile] [files]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **-f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **-e** option and no **-f** option, the flag **-e** may be omitted. The **-n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a /regular *expression*/ in the style of *ed*(1) modified thus:

1. In a context address, the construction *?regular expression?*, where **?** is any character, is identical to */regular expression/*. Note that in the context address **xabcxdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

2. The escape sequence **\n** matches a newline character *embedded* in the pattern space.

3. A period **.** matches any character except the *terminal* new-line character of the pattern space.

4. A command line with no addresses selects every pattern space.

5. A command line with one address selects each pattern space that matches the address.

6. A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with **** to hide the new-line character. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1)**a**

text — *Append*. Place *text* on the output before reading the next input line.

(2)**b** *label* — *Branch* to the : command bearing the *label*. If *label* is empty, branch to the end of the script.

(2)**c**

text — *Change*. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2)**d** — *Delete* the pattern space. Start the next cycle.

(2)**D** — *Delete* the initial segment of the pattern space through the first new-line character. Start the next cycle.

(2)**g** — *Replace* the contents of the pattern space by the contents of the hold space.

(2)**G** — *Append* the contents of the hold space to the pattern space.

(2)**h** — *Replace* the contents of the hold space by the contents of the pattern space.

(2)**H** — *Append* the contents of the pattern space to the hold space.

(1)**i**

text — *Insert*. Place *text* on the standard output.

(2)**l** — *List* the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2)**n** *Copy* the pattern space to the standard output. Replace the pattern space with the next line of input.

(2)**N** *Append* the next line of input to the pattern space with an embedded new-line character. (The current line number changes.)

(2)**p** *Print*. Copy the pattern space to the standard output.

(2)**P** *Copy* the initial segment of the pattern space through the first new-line character to the standard output.

(1)**q** *Quit*. Branch to the end of the script. Do not start a new cycle.

(2)**r** *rfile* *Read* the contents of *rfile*. Place them on the output before reading the next input line.

(2)**s**/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

g *Global*. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p *Print the pattern space if a replacement was made.*

w *wfile* *Write*. Append the pattern space to *wfile* if a replacement was made.

(2)**t** *label* *Test*. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of **t**. If *label* is empty, branch to the end of the script.

(2)**w** *wfile* *Write*. Append the pattern space to *wfile*.

(2)**x** *Exchange* the contents of the pattern and hold spaces.

(2)**y**/*string1*/*string2*/
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function* *Don't*. Apply the *function* (or group, if *function* is **{**) only to lines *not* selected by the address(es).

(0)**:** *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1)**=** *Place* the current line number on the standard output as a line.

(2)**{** Execute the following commands through a matching **}** only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO

awk(1), ed(1), grep(1).
User Guide

NAME

sh, rsh - shell, the standard/restricted command programming language

SYNOPSIS

sh [**-acefhiknrstuvx**] [args]
rsh [**-acefhiknrstuvx**] [args]

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non-blank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by | (or, for historical compatibility, by ^). The standard output of each command except the last one is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-line characters may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word ...*] **do** *list* **done**
Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word ...* is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ... **)** *list* **;;**] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see *Filename Generation* below).

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list is executed. If no* **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in

the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(list)
Execute *list* in a sub-shell.

{list;}
list is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments.

A word beginning with # causes that word and all the following characters up to a new line to be ignored.

Command Substitution.

The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing new lines are removed.

Parameter Substitution.

The character **$** is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

name=value [*name*=value] ...

Pattern-matching is not performed on *value*.

${parameter**}** A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *, @, #, ?, -, $, and !. The value, if any, of the parameter is substituted. The braces are required only when

parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is * or @, then all the positional parameters, starting with $1, are substituted (separated by spaces). Parameter $0 is set from argument zero when the shell is invoked.

${parameter:-word} If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${parameter:=word} If *parameter* is not set or is null, set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned in this way.

${parameter:?word} If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message **parameter null or not set** is printed.

${parameter:+word} If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above structures, *word* is not evaluated unless it is to be used as the substituted string; in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last synchronously executed command.

$ The process number of this shell.

! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution*, below). The user may not change **PATH** if executing under *rsh*.

CDPATH The search path for the *cd* command.

MAIL If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.

PS1 Primary prompt string, by default ``**$** ''.

PS2 Secondary prompt string, by default ``> ''.

IFS Internal field separators, normally **space**, **tab**, and **newline**.

SHACCT If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1) and *acctcms*(1M) can be used to analyze the data collected.

SHELL When the shell is invoked, it scans the environment (refer to **Environment** below) for this name. If it is found and

there is an **r** in the filename part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, and **IFS**, and **MAILCHECK**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation.

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation.

Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, then the word is left unchanged. The character . at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly.

`*` Matches any string, including the null string.

`?` Matches any single character.

`[...]` Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening ``['' is a ``!'' then any character not enclosed is matched.

Quoting.

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > newline space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\newline** is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occur and \ quotes the characters \, `, ", and **$**. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2"** ....

Prompting.

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline character is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output.

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple command or may precede or follow a command and are not passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document

becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter substitution and command substitution occur, (unescaped) **\newline** is ignored, and \ must be used to quote the characters \, **$**, `, and the first character of *word*. If - is appended to <<, all leading tabs are stripped from *word* and from the document.

<&digit The standard input is duplicated from file descriptor *digit* (see *dup*(2)). The standard output can be duplicated similarly, using > in place of <.

<&- The standard input is closed. The standard output can be closed similarly, using > in place of <.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to- right. For example,

... 1>xxx 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the

invoking shell, as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment.

The *environment* (see *environ*(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by an executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 cmd args

and

(export TERM; TERM=450; cmd args)

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution.

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a **/**, the search path is not used; such commands are not executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

Special Commands.

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

:	No effect; the command does nothing. A zero exit code is returned.
. *file*	Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.
break [*n*]	Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.

continue [*n*] — Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*] — Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / then the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

eval [*arg ...*] — The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg ...*] — The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*] — Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file also causes the shell to exit.)

export [*name ...*] — The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed.

newgrp [*arg* ...] Equivalent to **exec newgrp** *arg* ....

read [*name* ...] One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...] The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

set [**--aefhkntuvx** [*arg* ...]]

-a Mark variables that are modified or created for export.

-e Exit immediately if a command exits with a non-zero exit status.

-f Disable filename generation.

-h Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).

-k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

-n Read commands but do not execute them.

-t Exit after reading and executing one command.

-u Treat unset variables as an error when substituting.

-v Print shell input lines as they are read.

-x Print commands and their arguments as they are executed.

-- Do not change any of the flags; useful in setting **$1** to -.

Using **+** rather than **-** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$-**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given, the values of all names are printed.

shift [*n*] The positional parameters from **$n+1** ... are renamed **$1** ... If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test*(1) for usage and description.

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ... *arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and

by the commands it invokes. If *n* is 0 then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name*] — for each *name*, indicate how it would be interpreted if used as a command name.

ulimit [**-fp**] [*n*] — imposes a size limit of *n*.

- **-f** imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.
- **-p** changes the pipe size to *n* (UNIX System/RT only). If no option is given, **-f** is assumed.

umask [*nnn*] — The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*] — Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for and the return code is zero.

Invocation.

If the shell is invoked through *exec*(2) and the first character of argument zero is -, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; Note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file

containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c *string* If the **-c** flag is present then commands are read from *string*.

-s If the **-s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

-i If the **-i** flag is present or if the shell input and output are attached to a terminal, the shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

-r If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only.

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

1. changing directory (see *cd*(1))

2. setting the value of **$PATH**

3. specifying path or command names containing /

4. redirecting output (> and >>)

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), env(1), login(1), newgrp(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), profile(4), environ(5).

CAUTION

If a command is executed and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If the current directory or one above it is moved, **pwd** may not give the correct response. Use the **cd** command with a full pathname to correct this situation.

BUGS

If the **-c** and **-t** options are used together, the **-t** option will be ignored.

NAME

sleep - suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command under specified conditions, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(2), sleep(3C).

BUGS

Time must be less than 65536 seconds.

NAME

sort - sort and/or merge files

SYNOPSIS

sort [**-cmu**] [**-o***output*] [**-y***kmem*] [**-z***recsz*] [**-dfiMnr**] [**-btx**] [*+pos1* [*-pos2*]] [*files*]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if - is used as a filename or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire line, and order is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merge only; the input files are already sorted.

-u Unique; suppress all but one in each set of lines that have equal keys.

-o*output* The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **-o** and *output*.

-ykmem The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **-y0** is guaranteed to start with minimum memory. By convention, **-y** (with no argument) starts with maximum memory.

-zrecsz The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **-c** or **-m** options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules:

-d "Dictionary" order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

-f Fold lowercase letters into uppercase letters.

-i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

-M Compare as months. The first three non-blank characters of the field are folded to uppercase and compared so that "JAN" < "FEB" <...< "DEC". Invalid fields compare low to "JAN". The **-M** option implies the **-b** option (see below).

-n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional

decimal point, is sorted by arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation +pos1 -pos2 restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing -pos2 means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

-t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

-b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

Pos1 and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**. A starting position specified by **+m.***n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means **.0**, indicating the first character of the *m*+1st field. If the **b** flag is in effect, *n* is counted from the first non-blank in the *m*+1st field; **+m.0b** refers to the first non-blank character in the *m*+1st field.

A last position specified by -m.*n* is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means **.0**, indicating the last character of the *m*th field. If the **b** flag is in effect, *n* is counted from the last leading blank in the *m*+1st field; **-m.1b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of **infile** with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of **infile1** and **infile2**, placing the output in **outfile** and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of **infile1** and **infile2**, using the first non-blank character of the second field as the sort key:

```
sort -r +1.0b -1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort -t: +2n -3 /etc/passwd
```

Print the lines of the already sorted file **infile**, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

sort -um +2 -3 infile

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long) and for disorder discovered under option **-c**. When the last line of an input file is missing a new-line character, *sort* appends one, prints a warning message, and continues.

NAME

spell, hashmake, spellin, hashcheck - find spelling errors

SYNOPSIS

spell [**-v**] [**-b**] [**-x**] [**-l**] [**-i**] [+local_file] [files]
/usr/lib/spell/hashmake
/usr/lib/spell/spellin *n*
/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

Spell ignores most *troff*(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*, and other spellings, this option insists upon **-ise** in words like **standardise**.

Under the **-x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(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* follows the chains of all included files. Under the **-i** option, *spell* ignores

all chains of included files.

Under the +local_file option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, 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

EXAMPLE

To insert the word *mux* in the hashed spelling list */usr/lib/spell/hlista*, execute the following steps:

1. change directory:

   cd /usr/lib/spell

2. recreate the nine-digit codes for all the words in *hlista*:

   hashcheck hlista > hlista.decode

3. generate the nine-digit hash code for the new word and append it to the file *hlista.decode*:

   echo mux | hashmake >> hlista.decode

4. use the command *wc* to discover the number of lines, corresponding to number of hash codes currently contained in *hlista.decode*

   wc hlista.decode

5. sort *hlista.decode* and create the new *hlista* with *spellin* (where *linenum* is the number of lines in *hlista.decode*):

   sort hlista.decode | spellin linenum > hlista

SEE ALSO

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1).

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file

that is added to the hashed *spelling_list* via *spellin*.

The British spelling feature was done by an American.

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, the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

stty - set the options for a terminal

SYNOPSIS

stty [**-a**] [**-g**] [options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **-a** option, it reports all of the option settings; with the **-g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below is provided in *termio*(7). Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (**-parenb**) enable (disable) parity generation and detection.

parodd (**-parodd**) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *termio*(7)).

0 hang up phone line immediately.

50 75 110 134 150

200 300 600 1200

1800 2400 4800

9600 exta extb Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)

hupcl (-hupcl) hang up (do not hang up) a DATA-PHONE(Reg.) data set connection on last close.

hup (-hup) same as **hupcl (-hupcl)**.

cstopb (-cstopb) use two (one) stop bits per character.

cread (-cread) enable (disable) the receiver.

clocal (-clocal) assume a line without (with) modem control.

Input Modes

ignbrk (-ignbrk) ignore (do not ignore) break on input.

brkint (-brkint) signal (do not signal) INTR on break.

ignpar (-ignpar) ignore (do not ignore) parity errors.

parmrk (-parmrk) mark (do not mark) parity errors (see *termio*(7)).

inpck (-inpck) enable (disable) input parity checking.

istrip (-istrip) strip (do not strip) input characters to seven bits.

inlcr (-inlcr) map (do not map) NL to CR on input.

igncr (-igncr) ignore (do not ignore) CR on input.

icrnl (-icrnl) map (do not map) CR to NL on input.

iuclc (**-iuclc**) map (do not map) uppercase alphabetics to lowercase on input.

ixon (**-ixon**) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (**-ixany**) allow any character (only DC1) to restart output.

ixoff (**-ixoff**) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (**-opost**) post-process output (do not post-process output; ignore all other output modes).

olcuc (**-olcuc**) map (do not map) lowercase alphabetics to uppercase on output.

onlcr (**-onlcr**) map (do not map) NL to CR-NL on output.

ocrnl (**-ocrnl**) map (do not map) CR to NL on output.

onocr (**-onocr**) do not (do) output CRs at column zero.

onlret (**-onlret**) on the terminal NL performs (does not perform) the CR function.

ofill (**-ofill**) use fill characters (use timing) for delays.

ofdel (**-ofdel**) fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 select style of delay for carriage returns (see *termio*(7)).

nl0 nl1 select style of delay for line feeds (see *termio*(7)).

tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *termio*(7)).

bs0 bs1 select style of delay for backspaces (see *termio*(7)).

ff0 ff1 select style of delay for form feeds (see *termio*(7)).

vt0 vt1 select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (**-isig**) enable (disable) the checking of characters against the special control characters INTR and QUIT.

icanon (**-icanon**) enable (disable) canonical input (ERASE and KILL processing).

xcase (**-xcase**) canonical (unprocessed) upper/lowercase presentation.

echo (**-echo**) echo back (do not echo back) every character typed.

echoe (**-echoe**) echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (**-echok**) echo (do not echo) NL after KILL character.

lfkc (**-lfkc**) the same as **echok** (**-echok**); obsolete.

echonl (**-echonl**) echo (do not echo) NL.

noflsh (**-noflsh**) disable (enable) flush after INTR or QUIT.

stwrap (**-stwrap**) disable (enable) truncation of lines longer than 79 characters on a synchronous line.

stflush (**-stflush**) enable (disable) flush on a synchronous line after every *write*(2).

stappl (**-stappl**) use application mode (use line mode) on a synchronous line.

Control Assignments

control-char c set the control character to *c*, where *control-char* is **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **min** ,**or** time (**min** and **time** are used with **-icanon**; see *termio*(7)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "^**d**" is a **CTRL-d**); "^**?**" is interpreted as DEL and "^**-**" is interpreted as undefined.

line *i* set line discipline to *i* (0 < *i* < 127)

Combination Modes

evenp or **parity** enable **parenb** and **cs7**.

oddp enable **parenb**, **cs7**, and **parodd**.

-parity, **-evenp**, or

-oddp — disable **parenb**, and set **cs8**.

raw (**-raw** or **cooked**) — enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).

nl (**-nl**) — unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**-lcase**) — set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (**-LCASE**) — same as **lcase** (**-lcase**).

tabs (**-tabs** or **tab3**) — preserve (expand to spaces) tabs when printing.

ek — reset ERASE and KILL characters back to normal **#** and **@**.

sane — resets all modes to some reasonable values.

term — set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1), ioctl(2).
System Administration Utilities Reference Manual.

NAME

su - become superuser or another user

SYNOPSIS

su [-] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., superuser).

To use *su*, the appropriate password must be supplied (unless one is already superuser). If the password is correct, *su* executes a new shell with the user ID set to that of the specified user. To restore normal user ID privileges, type an **EOF** to the new shell.

Any additional arguments are passed to the shell, permitting the superuser to run shell procedures with restricted privileges (an *arg* of the form **-c** *string* executes *string* via the shell). When additional arguments are passed, **/bin/sh** is always used. When no additional arguments are passed, *su* uses the shell specified in the password file.

An initial **-** flag causes the environment to be changed to the one that would be expected if the user actually logged in again. This is done by invoking the shell with an *arg0* of **-su**, causing the **.profile** in the home directory of the new user ID to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for root. Note that the **.profile** can check *arg0* for **-sh** or **-su** to determine how it was invoked.

FILES

/etc/passwd system password file

$HOME/.profile user's profile

SEE ALSO

env(1), login(1), sh(1), environ(5).

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 prints the number of blocks in the file. It is typically used to look for bad spots or to validate a file communicated over a transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

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] [+m*n*] [-T*type*]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that these terminals behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the left margin on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **-8**, i.e., "standard" LSX tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even if the column markers begin at 0, as on the DASI 300, DASI 300s, and DASI 450.

-*code* Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:

-a 1,10,16,36,72
Assembler, IBM S/370, first format

-a2 1,10,16,40,72
Assembler, IBM S/370, second format

-c 1,8,12,16,20,55
COBOL, normal format

-c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **-c2**. This is the recommended format for COBOL. The appropriate format specification is:

<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

-n A repetitive specification requests tabs at columns 1+*n*, 1+2**n*, etc. Note that such a setting leaves a left margin of *n* columns on TermiNet terminals *only*. Of particular importance is the value **-8**: this represents the "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff* **-h** option for high-speed output. Another special case is the value **-0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

--file If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

```
tabs -- file; pr file
```

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-Ttype *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* searches for the **$TERM** value in the environment (see *environ*(5)). If no *type* can be found, *tabs* tries a sequence that works for many terminals.

+mn The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

DIAGNOSTICS

illegal tabs Arbitrary tabs are ordered incorrectly.

illegal increment A zero or missing increment is found in an arbitrary specification.

unknown tab code A "canned" code cannot be found.

can't open The --file option was used, and file can't be opened.

file indirection The --file option was used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

nroff(1), environ(5), term(5).
Advanced Utilities User Guide

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin. It is generally

impossible to usefully change the left margin without also setting tabs. *Tabs* clears only 20 tabs (on terminals requiring a long sequence), but can set 40 tabs.

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 does not terminate after the line of the input file has been copied, but enters 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

prints 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

prints 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(1).

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

tar - tape file archiver

SYNOPSIS

tar [key] [files]

DESCRIPTION

Tar saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function letter portion of the key is specified by one of the following:

r The named *files* are written on the end of the tape. The **c** function implies this function.

x The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t The names of the specified files are listed each time that they occur on the tape. If no *files* argument is given, all the names on the tape are listed.

u The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape.

c A new tape is created; writing begins at the beginning of the tape, instead of after the last file. This command implies the **r** function.

The following characters (function modifiers) may be used in addition to the letter that selects the desired function:

0,...,7 This modifier selects the drive on which the tape is mounted. The default is **0**.

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w This modifier causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no".

f This modifier causes *tar* to use the next argument as the name of the archive instead of **/dev/mt?**. If the name of the file is **-**, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

cd fromdir; tar cf - . | (cd todir; tar xf -)

b This modifier causes *tar* to use the next argument as the blocking factor for tape records. The default is 10, the maximum is 20. This option should be used only with raw magnetic tape archives (see **f** above). The block size is

determined automatically when reading tapes (key letters **x** and **t**).

l This modifier tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m This modifier tells *tar* to not restore the modification times. The modification time of the file is the time of extraction.

o This modifier causes extracted files to take on the user and group identifier of the user running *tar*, rather than those of the archived files. This modifier is valid only with the **x** option.

-C If a file name is preceded by **-C** in a **c** (create) or **r** (replace) operation, *tar* will perform a *chdir*(2) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from */usr/include* and from */etc*, one might use:

tar c -C /usr include -C /etc .

Note that the **-C** option applies to one following directory name and one following file name.

-M This modifier, used in combination with the **ctx** options, allows the manipulation of *tar* multivolume format files. Using this modifier, when a volume's EOF is reached the next volume's mount is requested of the operator in an interactive way.

FILES

/dev/mt?
/tmp/tar*

EXAMPLES

The following is a simple example using *tar* to create an archive of your home directory onto */dev/rmt8*. First position yourself in your home directory:

cd

create the archive:

tar cvf /dev/rmt8

Messages from *tar* will be displayed.

The **c** option means create the archive; the **v** option makes *tar* tell you what it is doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (*/dev/rmt8* in this example).

Now you can read the table of contents from the archive in the following way:

tar tvf /dev/rmt8

Messages from *tar* will be displayed.

Where the **t** option is for displaying the table-of-contents of the archive. You can extract files from the archive in the following way:

tar xvf /dev/rmt8

Messages from *tar* will be displayed.

DIAGNOSTICS

Complains about bad key characters and tape read/write errors.

Complains if enough memory is not available to hold the link tables.

SEE ALSO

Advanced Utilities User Guide

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on filename length is 100 characters.

NAME

tee - pipe fitting

SYNOPSIS

ltee [-i] [-a] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The:

-i ignores interrupts;

-a causes the output to be appended to the *files* rather than overwriting them.

NAME

test - condition evaluation command

SYNOPSIS

test expr
[expr **]**

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

-r *file*	true if *file* exists and is readable.
-w *file*	true if *file* exists and is writable.
-x *file*	true if *file* exists and is executable.
-f *file*	true if *file* exists and is a regular file.
-d *file*:	true if *file* exists and is a directory.
-c *file*	true if *file* exists and is a character special file.
-b *file*	true if *file* exists and is a block special file.
-p *file*	true if *file* exists and is a named pipe (fifo).
-u *file*	true if *file* exists and its set-user-ID bit is set.

-g *file* true if *file* exists and its set-group-ID bit is set.

-k *file* true if *file* exists and its sticky bit is set.

-s *file* true if *file* exists and has a size greater than zero.

-t [*fildes*] true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

-z *s1* true if the length of string *s1* is zero.

-n *s1* true if the length of the string *s1* is non-zero.

s1 **=** *s2* true if strings *s1* and *s2* are identical.

s1 **!=** *s2* true if strings *s1* and *s2* are *not* identical.

s1 true if *s1* is *not* the null string.

n1 **-eq** *n2* true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

! unary negation operator.

-a binary *and* operator.

-o binary *or* operator (**-a** has higher precedence than **-o**).

(expr) parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

SEE ALSO

find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses **[]**, rather than the word *test*), the square brackets must be delimited by blanks.

NAME

time - time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the time elapsed during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The execution time can depend on what kind of memory the program happens to land in; the user time in X/OS is often half what it is in core.

The times are printed on standard error.

SEE ALSO

times(2)

NAME

touch - update access and modification times of a file

SYNOPSIS

touch [**-amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch causes the access and modification times of each argument to be updated. If no time is specified (see *date*(1)) the current time is used. The **-a** and **-m** options cause *touch* to update only the access or modification times respectively (default is **-am**). The **-c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1), utime(2).

NAME

tput - query terminfo database

SYNOPSIS

tput [-Ttype] capname

DESCRIPTION

Tput uses the *terminfo*(*4*) database to make terminal- dependent capabilities and information available to the shell. *Tput* outputs a string if the attribute (**capability 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, 1 for FALSE) and does no output.

-Ttype indicates the type of terminal. Normally this flag is unnecessary because the default is taken from the environment variable **$TERM**.

Capname indicates the attribute from the *terminfo* database (refer to *terminfo*(*4*)).

EXAMPLES

tput clear	Echo clear-screen sequence for the current terminal.
tput cols	Print the number of columns for the current terminal.
tput -T450 cols	Print the number of columns for the 450 terminal.

bold=`tput smso` Set shell variable "bold" to stand-out mode sequence for current terminal. This might be followed by a prompt:

echo "${bold}Please type in your name: \c"

tput hc Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/usr/lib/terminfo/?/*

/etc/term/?/* Terminal descriptor files

/usr/include/term.h Definition files

/usr/include/curses.h

DIAGNOSTICS

Tput prints error messages and returns the following error codes on error:

-1 Usage error.

-2 Bad terminal type.

-3 Bad capname.

In addition, if a capname is requested for a terminal that has no value for that capname (e.g., **tput -T450 lines**), **-1** is printed.

SEE ALSO

stty(1), terminfo(4).

NAME

tr - translate characters

SYNOPSIS

tr [**-cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **-cds** may be used:

-c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

-d Deletes all input characters in *string1*.

-s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a-z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a*n] Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

The following example creates a list of all the words in *file1*, one per line, in *file2*. A word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for new line.

```
tr -cs "[A-Z][a-z]" "[ 12*]" < file1 > file2
```

SEE ALSO

ed(1), sh(1), ascii(5).

BUGS

Won't handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

NAME

true, false - provide truth values

SYNOPSIS

true
false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
   command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tty - get the terminal's name

SYNOPSIS

tty [-l] [-s]

DESCRIPTION

Tty prints the pathname of the user's terminal. The **-l** option prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line. The **-s** option inhibits printing of the terminal's pathname, allowing one to test just the exit code.

EXIT CODES

2 if invalid options were specified,

0 if standard input is a terminal,

1 otherwise.

DIAGNOSTICS

The message **"not on an active synchronous line"** means the standard input is not a synchronous terminal and the **-l** option is specified.

The message **"not a tty"** means the standard input is not a terminal and the **-s** option is not specified.

SEE ALSO

Advanced Utilities User Guide

NAME

type - interprets words as possible command names.

SYNOPSIS

type [*name*]

DESCRIPTION

For each *name* entered, *type* reports on how it would be interpreted if it were used as a command name.

SEE ALSO

sh(1).

NAME

umask - set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

NAME

uname - print name of current UNIX System

SYNOPSIS

uname [-snrvma]

DESCRIPTION

Uname prints the current system name of the UNIX System on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by *uname*(2) to be printed:

-s print the system name (default).

-n print the nodename (the nodename may be a name that the system is known by to a communications network).

-r print the operating system release.

-v print the operating system version.

-m print the machine hardware name.

-a print all the above information.

Arguments not recognized default the command to the **-s** option.

SEE ALSO

uname(2).

NAME

uniq - report repeated lines in a file

SYNOPSIS

uniq [**-udc** [**+n**] [**-n**]] [input [output]]

DESCRIPTION

Uniq reads the input file and compares adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-n The first *n* fields, together with any blanks before each, are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units - conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have:  inch
You want:  cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have:  15 lbs force/in2
You want:  atm
        * 1.020689e+00
        / 9.797299e-01
```

Units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with some less familiar units, and a few constants of nature, including:

pi ratio of circumference to diameter

c speed of light

e charge on an electron

g acceleration of gravity

force same as **g**

mole Avogadro's number

water pressure head per unit height of water

au astronomical unit

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

cat /usr/lib/unittab

FILES

/usr/lib/unittab

NAME

uucp, uulog, uuname - unix to unix copy

SYNOPSIS

uucp [options] source-files destination-file
uulog [options]
uuname [**-l**] [**-v**]

DESCRIPTION

Uucp.

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on your machine, or may have the form:

system-name!pathname

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!*system-name*!...!*system-name*!pathname

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are willing to forward information.

The shell metacharacters **?**, ***** and **[...]** appearing in *pathname* are expanded on the appropriate system.

Pathnames may be:

1. A full pathname

2. A pathname preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory

3. A pathname preceded by *~/user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR Anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system, the copy fails. If the *destination-file* is a directory, the last part of the *source-file* name is used.

Uucp preserves execute permissions across the transmissipn and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*:

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-c Use the source file when copying out rather than copying the file to the spool directory (default).

-C Copy the source file to the spool directory.

-mfile Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

-nuser Notify *user* on the remote system that a file was sent.

-esys Send the *uucp* command to system *sys* to be executed there. (Note: this is only successful if the remote machine allows the *uucp* command to be executed by **/usr/lib/uucp/uuxqt**.)

-r Queue job but do not start the file transfer process. By default a file transfer process is started each time *uucp*

is invoked.

-j Control writing of the *uucp* job number to standard output (see below).

Uucp associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate the job.

The environment variable **JOBNO** and the **-j** option are used to control the listing of the *uucp* job number on standard output. If the environment variable **JOBNO** is undefined or set to **OFF**, the job number will not be listed (default). If *uucp* is then invoked with the **-j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time *uucp* is invoked. In this case, the **-j** option will suppress output of the job number.

Uulog.

Uulog queries a summary log of *uucp* and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE.**

The options cause *uulog* to print logging information:

-ssys Print information about work involving system *sys*. If *sys* is not specified, then logging information for all systems will be printed.

-uuser Print information about work done for the specified *user*. If *user* is not specified, then logging information for all suers will be printed.

Uuname.

Uuname lists the *uucp* names of known systems. The **-l** option returns the local system name. The **-v** option prints additional information about each system. A description is printed for each system that has a line of information in **/usr/lib/uucp/ADMIN**. The format of

ADMIN is: *sysname***tab**description**tab**.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uux(1C), chmod(2).

CAUTION

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You are probably not able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you are probably not able to send files to arbitrary pathnames. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to **~nuucp** or just **~**).

In order to send files that begin with a dot (e.g., **.profile**), the files must be qualified with a dot. For example, **.profile**, **.prof***, and **.profil?** are correct, while ***prof*** and **?profile** are incorrect.

BUGS

All files received by *uucp* are then owned by *uucp*. The **-m** option only works for sending files or receiving a single file. Receiving multiple files specified by special shell characters **? * [...]** does not activate the **-m** option. The **-m** option does not work if all transactions are local or if *uucp* is executed remotely via the **-e** option. The **-n** option functions only when the source and destination are not on the same machine.

Only the first six characters of a *system-name* are significant. Any excess characters are ignored.

NAME

uustat - uucp status inquiry and job control

SYNOPSIS

uustat [options]

DESCRIPTION

Uustat displays the status of, or cancels, previously specified *uucp* commands, or provides general status on *uucp* connections to other systems. The following *options* are recognized:

-**j**jobn Report the status of the *uucp* request *jobn*. If **all** is used for *jobn*, the status of all *uucp* requests is reported. If *jobn* is omitted, the status of the current user's *uucp* requests is reported.

-**k**jobn Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the superuser.

-**r**jobn Rejuvenate *jobn*. *Jobn* is touched so that its modification time is set to the current time. This prevents *uuclean* from deleting the job until the jobs modification time reaches the limit imposed by *uuclean*.

-**c**hour Remove status entries older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the superuser.

-**u**user Report the status of all *uucp* requests issued by *user*.

-**s**sys Report the status of all *uucp* requests that communicate with remote system *sys*.

-ohour Report the status of all *uucp* requests which are older than *hour* hours.

-yhour Report the status of all *uucp* requests which are younger than *hour* hours.

-mch Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* is provided.

-Mch This is the same as the *-m* option except that two times are printed: the time that the last status was obtained and the time that the last successful transfer to that system occurred.

-O Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each *uucp* request.

-q List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **-j**, **-m**, **-k**, **-c**, **-r**, can be used with the rest of the other options.

For example, the command:

uustat -uhdc -smhtsa -y72

prints the status of all *uucp* requests that were issued by user **hdc** to communicate with system **mhtsa** within the last 72 hours. The job request status statement provides the following information:

status
job-number user remote-system command-time status-time

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
000001	the copy failed, but the reason cannot be determined
000002	permission to access local file is denied
000004	permission to access remote file is denied
000010	bad *uucp* command is generated
000020	remote system cannot create temporary file
000040	cannot copy to remote directory
000100	cannot copy to local directory
000200	local system cannot create temporary file
000400	cannot execute *uucp*
001000	copy (partially) succeeded
002000	copy finished, job deleted
004000	job is queued
010000	job killed (incomplete)
020000	job killed (complete)

The format of the machine accessibility status statement is:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/L_stat	system status file
/usr/lib/uucp/R_stat	request status file

SEE ALSO

uucp(1C).

NAME

uuto - public UNIX System-to-UNIX System file copy

SYNOPSIS

uuto [options] source-files destination

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 pathname on the user's machine. Destination has the form:

system!user

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *Logname* is the login name of someone on the specified system.

Two *options* are available:

-p Copy the source file into the spool directory before transmission.

-m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/*user***/***mysystem***/***files*

where the full path of PUBDIR is **/usr/spool/uucppublic**, *user* is the

login name of the recipient, and *mysystem* is the system name of the source.

The destined recipient is notified by *mail*(1) of the arrival of files.

FILES

PUBDIR	/usr/spool/uucppublic	public directory

SEE ALSO

mail(1), uuclean(1M), uucp(1C), uustat(1C), uux(1C).
Advanced Utilities User Guide

NAME

uux - UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

Uux gathers zero or more files from various systems, executes a command on a specified system and then sends standard output to a file on a specified system. Note that, for security reasons, many installations limit the list of commands executable on behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail (see *mail*(1)) via *uux*.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command names and filenames may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

Filenames may be:

1. A full pathname
2. A pathname preceded by ~xxx where *xxx* is a login name on the specified system and is replaced by that user's login directory
3. Prefixed by the current directory

As an example, the command

uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"

gets the **f1** files from the **usg** and **pwba** machines, executes a *diff* command and puts the results in **f1.diff** in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or by quoting the special characters as individual arguments.

Uux attempts to get all files to the execution system. For output files, the filename must be escaped using parentheses. For example, the command

uux a!uucp b!/usr/file \(c!/usr/file\)

sends a *uucp* command to system "a" to get **/usr/file** from system "b" and send it to system "c".

Uux notifies you if the requested command on the remote system is disallowed. The response comes by remote mail from the remote machine. Executable commands are listed in **/usr/lib/uucp/L.cmds** on the remote system. The format of the **L.cmds** file is:

cmd,machine1,machine2,...

If no machines are specified, then any machine can execute *cmd*. If machines are specified, only the listed machines can execute *cmd*. If the desired command is not listed in **L.sys**, then no machine can execute that command.

Redirection of standard input and output is usually restricted to files in **PUBDIR**. Directories into which redirection is allowed must be specified in **/usr/lib/uucp/USERFILE** by the system administrator.

The following options are interpreted by *uux*:

- The standard input to *uux* is made the standard input to the *command-string*.

-n Send no notification to user.

-■file Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

-j Control writing of the *uucp* job number to standard output.

Uux associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate a job.

The environment variable **JOBNO** and the **-j** option are used to control the listing of the *uux* job number on standard output If the environment variable **JOBNO** is undefined or set to **OFF**, the job number will not be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time *uux is invoked. In this case, the* **-j** option will suppress output of the job number.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public director (PUBDIR)
/usr/lib/uucp/*	other data and programs

SEE ALSO

mail(1), uuclean(1M), uucp(1C).
Advanced Utilities User Guide

BUGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

Only the first six characters of the *system-name* are significant. Any excess characters are ignored.

NAME

vi - screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**-t** tag] [**-r** file] [**-l**] [**-w**n] [**-R**] [+command] name ...
view [**-t** tag] [**-r** file] [**-l**] [**-w**n] [**-R**] [+command] name ...
vedit [**-t** tag] [**-r** file] [**-l**] [**-w**n] [**-R**] [+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. See "The Editors" in the *X/OS User Manual* for full details on using *vi*.

INVOCATION

The following invocation options are interpreted by *vi*:

-ttag — Edit the file containing the *tag* and position the editor at its definition.

-rfile — Recover *file* after an editor or system crash. If *file* is not specified, a list of all saved files is printed.

-l — LISP mode; indents appropriately for lisp code. The **()** **{}** **[[** and **]]** commands in *vi* and *open* are modified to have meaning for *lisp*.

-wn 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.

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.

NOTE: In the following tables, the notation **CR** indicates carriage returns that the user must enter. **ESC** indicates the key marked **ESCAPE** or **ESC** key.

VI MODES

Command Normal and initial mode. Other modes return to command state upon completion. **ESC** (escape) is used to cancel a partial command.

Input Entered by **a i A I o O c C s S R**. Arbitrary text may then be entered. Input mode is normally 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
hjkl	same as arrow keys
i*abc***ESC**	insert text *abc*
cw*new***ESC**	change word to *new*
esESC	pluralize word
x	delete a character
dd	delete a line
3dd	delete 3 lines
dw	delete a word
xp	transpose characters
u	undo previous change
ZZ	exit *vi*, saving changes
:q!	quit, discarding changes
/text/	search for *text*
:*cmd*	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 the following ways:

line/column number	**z G |**
scroll amount	**^D ^U**

repeat effect most of the rest

Interrupting, canceling

ESC end insert or incomplete cmd

^? (delete or rubout) interrupts

^L reprint screen if ^? scrambles it

^R reprint screen if ^L is key

File manipulation

:w	write back changes
:wq	write and quit
:q	quit
:q!	quit, discard changes
:e *name*	edit file *name*
:e!	reedit, discard changes
:e + *name*	edit, starting at end
:e +*n*	edit starting at line *n*
:e #	edit alternate file
^	synonym for :e #
:w *name*	write file *name*

:w! *name*	overwrite file *name*
:sh	run shell, then return
:!*cmd*	run *cmd*, then return
:n	edit next file in arglist
:n *args*	specify new arglist
:f	show current line as percent of file
^G	synonym for :f
:ta *tag*	tag file entry *tag*
^]	:ta, following word is *tag*

In general, any *ex* or *ed* command 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	goto line (end default)
/pat	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?

N	reverse last / or ?
/pat/+n	*n*'th line after *pat*
?pat?-n	*n*'th 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
z	redraw, current at window top
z-	redraw, current at bottom
z.	redraw, current at center
/pat/z-	*pat* line at bottom
z*n*	use *n*-line window
^E	scroll window down 1 line

^Y	scroll window up 1 line

Marking and returning

``	move cursor to previous context
''	move cursor to first non-white space in line
m*x*	mark position with letter *x*
`*x*	to mark *x*
'*x*	move cursor to first non-white space 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 space on line
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*	up to *x* forward
T*x*	back up to *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

Commands for LISP Mode

)	Forward s-expression
}	... but do not stop at atoms
(	Back s-expression
{	... but don't stop at atoms

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as ^H
kill	your kill, erase input this line
\	escapes ^H, your erase and kill

ESC	ends insertion, back to command
^?	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
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 (double to affect lines)

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
<	left shift
>	right shift
!	filter through command
=	indent for LISP
y	yank lines to buffer

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

The "put" command 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 text 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

SEE ALSO

ex (1).
The Editors in the *X/OS User Manual*.

WARNINGS AND BUGS

Software tabs using ^T work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this

correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open*.

NAME

wait - await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & have completed and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

BUGS

This command cannot wait for processes of a 3-or-more-stage pipeline that are not children of the shell.

NAME

wc - word count

SYNOPSIS

wc [**-lwc**] [names]

DESCRIPTION

Wc counts lines, words and characters in the named files or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-line characters.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **-lwc**.

When *names* are specified on the command line, they are printed along with the counts.

NAME

who - who is on the system

SYNOPSIS

who [**-uTHlpdbrtasq**] [file]
who am i

DESCRIPTION

Who lists the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current system user. It examines the **/etc/utmp** file to obtain its information. If *file* is given, that file is examined. Usually, *file* is **/etc/wtmp**, which contains a history of all the logins since the file was last created.

Who with the **am i** or **am I** option identifies the invoking user.

Except for the default **-s** option, the general format for output entries is:

name [*state*] *line time activity pid* [*comment*] [*exit*]

With options, *who* lists logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

-u List information about users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity

in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This file can contain information such as where the terminal is located, the telephone number of the dataset, or type of terminal if hard-wired.

-T This option is the same as the **-u** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A **+** appears if the terminal is writable by anyone; a **-** appears if it is not. **Root** can write to all lines having a **+** or a **-** in the *state* field. If a bad line is encountered, a **?** is printed.

-l List only those lines on which the system is waiting for someone to log in. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field doesn't exist.

-H This option prints column headings above the regular output.

-q This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

-p List any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

-d Display all processes that have expired and have not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(2)), of the dead process. This can be useful in determining why a process terminated.

-b Indicate the time and date of the last reboot.

-r Indicate the current *run-level* of the *init* process. Following the run-level and date information are three fields which indicate the current state, the number of times that state was previously entered, and the previous state.

-t Indicate the last change to the system clock (via the *date*(1) command) by **root**. See *su*(1).

-a Process **/etc/utmp** or the named *file* with all options turned on.

-s List only the *name*, *line*, and *time* fields; this is the default.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

init(1M) in the *System Administration Reference Manual*.
date(1), login(1), mesg(1), su(1), wait(2), inittab(4), utmp(4).
Advanced Utilities User Guide

NAME

write - write to another user

SYNOPSIS

write user [line]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* **(tty??)** [*date*]...

to your intended recipient. When it has successfully completed the connection, it sends two bells to your terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal is to be connected (e.g., **tty00**); otherwise, the first instance of the user found in **/etc/utmp** is assumed and the following message is posted:

user **is logged on more than one place.**
You are connected to *terminal*.
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg*(*1*) command. Writing to others is normally allowed by default. Certain

commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent interference with their output; however, if the user has superuser permissions, messages can be forced onto a write-inhibited terminal.

If the character ! is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp to find user

/bin/sh to execute !

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), who(1).
Advanced Utilities User Guide

DIAGNOSTICS

user **not logged in** means the person you are trying to *write* to is not logged in.

NAME

xmt, rcv - send/receive files to/from an MS-DOS computer

SYNOPSIS

/usr/rl/xmt.v5 *[options] ... file*
/usr/rl/rcv.v5 *[options] ... file*

DESCRIPTION

Xmt and **rcv** are two commands to implement a file transfer between a UNIX machine and an MS-DOS machine.
They are invoked by two corresponding programs running on the MS-DOS machine, that take care to pass the correct arguments. These MS-DOS programs are normally part of the OLINET-LAN X/OS SERVER - VT220 EMULATOR package.

Xmt and **rcv** should NOT be called directly by the user, as they use standard input and output to talk with the MS-DOS programs and to transfer files.

The following options are recognised:

-b:*n* Uses *n* bytes packets to send or receive data. *n* must be between 64 and 256. Default is 128.

-c Compress (**xmt**) or decompress (**rcv**) data to speed up the transfer of large files.

-u Send file name before each file to allow local shell file name expansion (for **xmt** only).

Rcv also recognises the following options:

-f Puts the received data into the named file, creating it if necessary.

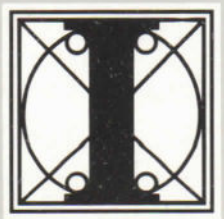
-p Pipe received data to the line printer spooler.

Either the **-f**, **-c** or **-p** option must be specified (as the last option) for **rcv**.

NOTE

If you inadvertently invoke one of these commands, you can cancel it by typing DEL (Ctrl-?).

Code 4041460 V (0)
Printed in Italy



olivetti