sun
microsystems

Commands Reference Manual
for the Sun Workstation

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1172-01
Revision E of 15 May, 1985

Trademarks

Multibus is a trademark of Intel Corporation.

Sun Workstation is a trademark of Sun Microsystems Incorporated.

UNIX is a trademark of Bell Laboratories.

Revision History

Revision	Date	Comments
A	23rd February 1983	First release of this Manual.
B	15th April 1983	Second Release of this manual involved many corrections to manual pages.
C	1st August 1983	Third Release of this manual involved many corrections to manual pages.
D	1st November 1983	Fourth Release of this manual involved many corrections to manual pages. Added OPTIONS to manual pages for clarity. Fixed numerous incorrect cross-references.
E	7 January 1984	Fifth Release of this manual involved many corrections to manual pages. The manual was renamed the *Commands Reference Manual* instead of the *User's Manual.*
F	15 May 1985	Corrected numerous bugs. Brought the *Maintenance Commands and Procedures* (formerly section 8) back into this manual and removed same from the *System Manager's Manual.* Page numbering is contiguous throughout the manual and we replaced the permuted index with a human-oriented index.

NAME

intro – introduction to commands

DESCRIPTION

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

(1) Commands of general utility.

(1C) Commands for communication with other systems.

(1G) Commands used primarily for graphics and computer-aided design.

SEE ALSO

- Section 6 in this manual for computer games.
- Section 7 in this manual for descriptions of publicly available files and macro packages for document preparation.
- Section 8 in this manual for system administration procedures, system maintenance and operation commands, plus descriptions of network services daemons and servers.
- *Getting Started* in the *Beginner's Guide to the Sun Workstation.*

DIAGNOSTICS

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

NAME

adb – debugger

SYNOPSIS

adb [**–w**] [**–k**] [**–I**dir] [objfil [corfil]]

DESCRIPTION

Adb is an interactive, general purpose debugger. It examines files and provides a controlled environment for the execution of UNIX programs.

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

OPTIONS

–w Create both *objfil* and *corfil* if necessary and open them for reading and writing so that files can be modified using *adb*.

–k Do UNIX kernel memory mapping; should be used when *core* is a UNIX crash dump or */dev/mem.*

–I specifies a directory where files to be read with $< or $<< (see below) will be sought; the default is */usr/lib/adb*.

USING ADB

Adb reads commands from the standard input and displays responses on the standard output. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

Adb saves and restores terminal characteristics when running a sub–process. This makes it possible to debug programs that manipulate the screen. See *tty*(4).

In general, requests to *adb* are of the form

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

The symbol *dot* (**.**) represents the current location. It is initially zero. If *address* is present then *dot* is set to *address*. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

EXPRESSIONS

. The value of *dot*.

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

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

& The last *address* typed. (Used to be “"”.)

integer A number. The prefixes 0o and 0O (“zero oh”) force interpretation in octal radix; the prefixes 0t and 0T force interpretation in decimal radix; the prefixes 0x and 0X force interpretation in hexadecimal radix. Thus 0o20 = 0t16 = 0x10 = sixteen. If no prefix appears, then the *default radix* is used; see the $d command. -The default radix is initially hexadecimal. The hexadecimal digits are 0123456789abcdefABCDEF with the obvious values. Note that if a hexadecimal number starts with a letter, but does not duplicate a defined symbol, it is accepted as a hexadecimal value. To enter a hexadecimal number that is the same as a defined symbol, preceed it by **0**, **0x**, or **0X**.

integer.fraction
A 32 bit floating point number.

´cccc´ The ASCII value of up to 4 characters. A backslash (‘\’) may be used to escape a ´.

< *name*
The value of *name*, which is either a variable name or a register name. *Adb* maintains a

number of variables (see VARIABLES) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil*. The register names are those printed by the $r command.

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

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

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

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

Unary operators

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

%*exp* The contents of the location addressed by *exp* in *objfil*. (Used to be @.)

−*exp* Integer negation.

~*exp* Bitwise complement.

#*exp* Logical negation.

^F*exp* (Control−f) Translates program addresses into sourcefile addresses. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

^A*exp* (Control−a) Translates sourcefile addresses into program addresses. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

`*name* (Back−quote) Translates a procedure name into a sourcefile address. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

"*filename*"
A filename enclosed in quotation marks (for instance, "main.c") produces the sourcefile address for the zero-th line of that file. Thus to reference the third line of the file main.c, we say: "main.c"+3. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

Binary operators are left associative and are less binding than unary operators.

e1+*e2* Integer addition.

e1−*e2* Integer subtraction.

e1 * *e2* Integer multiplication.

e1%*e2* Integer division.

e1&*e2* Bitwise conjunction.

e1 |*e2* Bitwise disjunction.

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

VARIABLES

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

0 The last value printed.

1 The last offset part of an instruction source.
2 The previous value of variable 1.
9 The count on the last $< or $<< command.

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

b The base address of the data segment.
d The data segment size.
e The entry point.
m The 'magic' number (0407, 0410 or 0413).
s The stack segment size.
t The text segment size.

COMMANDS

Adb commands consist of a **verb** followed by a **modifier** or list of modifiers.

The verbs are:

? Print locations starting at *address* in *objfil*.

/ Print locations starting at *address* in *corfil*.

= Print the value of *address* itself.

@ Interpret *address* as a sourcefile address, and print locations in *objfile* or lines of the source text. (Works only if the program has been compiled using the **–go** flag. See *cc*(1).)

: Manage a subprocess.

$ Execute miscellaneous commands.

> Assign a value to a variable or register.

RETURN
Repeat the previous command with a *count* of 1. *Dot* is incremented by its current increment.

! Call the shell to execute the following command.

Each verb has a specific set of **modifiers**, these are described below.

[?, /, @, =] Modifiers

The first four verbs described above take the same set of modifiers. These modifiers specify the *format* of command output. Each modifier consists of a letter preceded by an optional repeat count; each verb may take one or more modifiers:

[**?**, **/**, **@**, **=**] [[*rcount*] **fletter ...**]

Each time one of these commands is given, *dot* is incremented by a certain amount (sum of the increments specific to each format letter, see below). If a command is given without a modifier, the last specified format is used to display output. The following lists the format letter, the amount *dot* increments each time the letter is used, and a description of what each letter does.

o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O 4 Print 4 bytes in octal.
q 2 Print in signed octal.
Q 4 Print long signed octal.
d 2 Print in decimal.
D 4 Print long decimal.

x 2 Print 2 bytes in hexadecimal.

X 4 Print 4 bytes in hexadecimal.

u 2 Print as an unsigned decimal number.

U 4 Print long unsigned decimal.

f 4 Print the 32 bit value as a floating point number.

F 8 Print double floating point.

b 1 Print the addressed byte in octal.

c 1 Print the addressed character.

C 1 Print the addressed character using the standard escape convention: print control characters as ^X and the delete character as ^?.

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

S *n* Print a string using the ^*X* escape convention (see **C** above). *n* is the length of the string including its zero terminator.

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

i n Print as machine instructions. *n* is the number of bytes occupied by the instruction. In this format, variables 1 and 2 are set to the offset parts of the source and destination respectively.

z n Print as machine instructions with 68010 instruction timings. *n* is the number of bytes occupied by the instruction. In this format, variables 1 and 2 are set to the offset parts of the source and destination respectively.

I 0 Print the source text line specified by *dot* (@ command) or most closely corresponding to *dot* (? command).

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

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

p 4 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.

A 0 Print the value of *dot* in sourcefile symbolic form, that is: "*filename*"+*nnn*. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

P 4 Print the addressed value in sourcefile symbolic form, that is: "*filename*"+*nnn*. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

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

r 0 Print a space.

n 0 Print a newline.

"..." 0 Print the enclosed string.

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

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

− *Dot* is decremented by 1. Nothing is printed.

[?, /] Modifiers

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

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

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

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

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

: Modifiers

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

B*c* Like *b* but takes a sourcefile address. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

d Delete breakpoint at *address*.

D Like *d* but takes a sourcefile address. (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

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

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

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

S Like **s** but single steps source lines, rather than machine instructions. This is acheived by repeatedly single−stepping machine instructions until the corresponding sourcefile address changes. (Thus procedure calls cause stepping to stop.) (Works only if the program has been compiled using the **−go** flag. See *cc*(1).)

i Add the signal specified by *address* to the list of signals which are passed directly to the subprocess with the minimum of interference. Normally, *adb* intercepts all signals destined for the subprocess, and the user must use the **:c** command to continue the process with the signal. Signals on this list are handed to the process with an implicit **:c***s* as soon as they are seen.

t Remove the signal specified by *address* from the list of signals that are implicitly passed to the subprocess.

k Terminate the current subprocess, if any.

$ Modifiers

<*file* Read commands from the file *file*. If this command is executed in a file, further commands in the file are not seen. If *file* is omitted, the current input stream is terminated. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable *9* before the first command in *file* is executed.

<<*file* Similar to <, but can be used in a file of commands without closing the file. Variable *9* is saved during the execution of this command, and restored when it

completes. There is a (small) finite limit to the number of **<<** files that can be open at once.

>file Append output to *file*, which is created if it does not exist. If *file* is omitted, output is returned to the terminal.

? Print process id, the signal which stopped the subprocess, and the registers. Produces the same response as **$** used without any modifier.

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

b Print all breakpoints and their associated counts and commands.

c C stack backtrace. If *address* is given then it is taken as the address of the current frame instead of the contents of the frame—pointer register. If *count* is given then only the first *count* frames are printed.

C Similar to **c**, but in addition prints the names and (32 bit) values of all automatic and static variables for each active function. (Works only if the program has been compiled using the **—go** flag. See *cc*(1).)

d Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus "10$d" never changes the default radix. To make decimal the default radix, use "0t10$d".

e Print the names and values of external variables.

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

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

o All integers input are regarded as octal.

q Exit from *adb*.

v Print all non zero variables in octal.

m Print the address map.

f Print a list of known source file names.

p Print a list of known procedure names.

P (*Kernel debugging*) Change the current kernel memory mapping to map the designated **user structure** to the address given by the symbol *_u*. The *address* argument is the address of the user's **proc** structure.

i Show which signals are passed to the subprocess with the minimum of *adb* interference. Signals may be added to or deleted from this list using the **:i** and **:t** commands.

W Re-open *objfile* and *corfile* for writing, as though the **—w** command—line argument had been given.

ADDRESS MAPPING

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged, addresses are interpreted in the usual way (as described below) in the address space of the subprocess. If the operating system is being debugged, either post-mortem or by using the special file **/dev/mem** to interactively examine and/or modify memory, the maps are set to map the kernel virtual addresses which start at zero. For some commands, the address is not interpreted as a memory address at all, but as an ordered pair representing a file number and a line number within that file. The @ command always takes such a sourcefile address, and several operators are available to convert to and from the more customary core locations.

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

$b1 \leq address < e1 \Rightarrow file\ address = address + f1 - b1$, otherwise, *b2≤address<e2 => file address=address+f2−b2,*

otherwise, the requested *address* is not legal. If a **?** or **/** is followed by an ***** then only the second triple is used.

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

FILES

a.out
core

SEE ALSO

cc(1), dbx(1), ptrace(2), a.out(5), core(5)
Using adb to debug the UNIX kernel in the *Sun System Internals Guide.*

DIAGNOSTICS

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

BUGS

There doesn't seem to be any way to clear all breakpoints.

Adb uses the symbolic information in an old and now obsolete format generated by the **−go** flag of *cc*(1); it should be changed to use the new format used by the *dbx* debugger and generated by **−g.**

Since no shell is invoked to interpret the arguments of the **:r** command, the customary wild−card and variable expansions cannot occur.

Since there is little type−checking on addresses, using a sourcefile address in an inappropriate context may lead to unexpected results: *'main?i* will almost certainly not do anything useful.

NAME

addbib – create or extend bibliographic database

SYNOPSIS

addbib [**−p** promptfile] [**−a**] database

DESCRIPTION

When this program starts up, answering "y" to the initial "Instructions?" prompt yields directions; typing "n" or RETURN skips them. *Addbib* then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to a *database*. A null response (just RETURN) means to leave out that field. A minus sign (−) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating "Continue?" prompt allows the user either to resume by typing "y" or RETURN, to quit the current session by typing "n" or "q", or to edit the *database* with any system editor *(vi, ex, edit, ed).*

The −a option suppresses prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL-d. The −p option causes *addbib* to use a new prompting skeleton, defined in *promptfile.* This file should contain prompt strings, a tab, and the key-letters to be written to the *database.*

The most common key-letters and their meanings are given below. *Addbib* insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

%A	Author's name
%B	Book containing article referenced
%C	City (place of publication)
%D	Date of publication
%E	Editor of book containing article referenced
%F	Footnote number or label (supplied by *refer*)
%G	Government order number
%H	Header commentary, printed before reference
%I	Issuer (publisher)
%J	Journal containing article
%K	Keywords to use in locating reference
%L	Label field used by −k option of *refer*
%M	Bell Labs Memorandum (undefined)
%N	Number within volume
%O	Other commentary, printed at end of reference
%P	Page number(s)
%Q	Corporate or Foreign Author (unreversed)
%R	Report, paper, or thesis (unpublished)
%S	Series title
%T	Title of article or book
%V	Volume number
%X	Abstract — used by *roffbib,* not by *refer*
%Y,Z	ignored by *refer*

Except for 'A', each field should be given just once. Only relevant fields should be supplied. An example is:

```
%A      Bill Tuthill
%T      Refer — A Bibliography System
%I      Computing Services
%C      Berkeley
%D      1982
```

%O UNX 4.3.5.

FILES

promptfile optional file to define prompting

SEE ALSO

refer(1), sortbib(1), roffbib(1), indxbib(1)

NAME

adjacentscreens – notify the window driver of the physical relationships of screens

SYNOPSIS

adjacentscreens [**–c** ¦ **–m**] *center screen* [[**–l** ¦ **–r** ¦ **–t** ¦ **–b**] *side screen*] [**–x**]

DESCRIPTION

Adjacentscreens tells the mouse cursor tracking mechanism of the window driver how to move between screens that contain windows. Once properly notified using *adjacentscreens*, the mouse cursor slides from one screen to another when the user moves the cursor off the edge of a screen.

OPTIONS

–c *center screen*

The *center screen* is a frame buffer device name, such as */dev/fb*. All the other physical screen positions are relative to this reference point. The **–c** flag (*c* for *c*enter) is optional. If no further arguments are present on the command line, *center screen* is set to have no neighbors.

–m *center screen*

The **–m** flag (*m* for *m*iddle) may be used instead of **–c**.

–l *side screen*

The *side screen* is also a frame buffer device name, such as */dev/cgone0*. The **–l** flag means that *side screen* is to the *l*eft of *center screen*. Up to four repetitions of "**flag** *side screen*" may be specified on the command line to define the four neighbors of *center screen*.

–r *side screen*

Like **–l**, but means that *side screen* is to the *r*ight of *center screen*.

–t *side screen*

Like **–l**, but means that *side screen* is on *t*op of *center screen*.

–b *side screen*

Like **–l**, but means that *side screen* is *b*elow *center screen*.

–x　Suppresses the normal notification to a *side screen* cursor tracker that *center screen* is its only neighbor. This option is useful if you have a large number of screens or want strange inter-window cursor movement.

EXAMPLE

A common configuration would be two screens, a monochrome (*/dev/fb*) and a color screen (*/dev/cgone0*). Let us assume that the user has set up an instance of *suntools* on each screen (the window systems **must** be running before *adjacentscreens* is run). He would notify the window driver that the color screen was to the right of the monochrome screen by running "% adjacentscreens /dev/fb -r /dev/cgone0" in a Shelltool (see *suntools*(1)). This sets up cursor tracking so that the cursor slides from the monochrome screen to the color screen when the cursor moves off the right hand side of the monochrome screen. Similarly, the cursor slides from the color screen to the monochrome screen when the cursor moves off the left hand side of the color screen.

FILES

/usr/suntool/adjacentscreens

SEE ALSO

suntools(1), login(1)

BUGS

Window systems on the screens have to be initialized before running *adjacentscreens*.

NAME

admin — create and administer SCCS files

SYNOPSIS

/usr/sccs/admin [**−n**] [**−i**[*name*]] [**−r***rel*] [**−t**[*name*]] [**−f***flag* [*flag-val*]] ...
[**−d***flag* [*flag-val*]] ... [**−a***login*] ... [**−e***login*] ... [**−m**[*mrlist*]]
[**−y**[*comment*]] [**−h**] [**−z**] *filename* ...

DESCRIPTION

Admin creates new SCCS files and changes parameters of existing ones. Options and SCCS file names may appear in any order on the *admin* command line. SCCS file names must begin with the characters '**s.**'. A named file is created if it doesn't exist already, and its parameters are initialized according to the specified options. Any parameter not initialized by an option is assigned a default value. If a named file does exist, parameters corresponding to specified options are changed, and other parameters are left as is.

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

OPTIONS

Options are explained as though only one named file is to be processed, since options apply independently to each named file.

−n A new SCCS file is being created.

−i[*name*]
Initial text: the file *name* contains the text of a new SCCS file. The text is the first delta of the file — see **−r** option for delta numbering scheme. If *name* is omitted, the text is obtained from the standard input. Omitting the **−i** option altogether creates an empty SCCS file. You can only create one SCCS file with an **admin −i** command. Creating more than one SCCS file with a single *admin* command requires that they be created empty, in which case the **−i** option should be omitted. Note that the **−i** option implies the **−n** option.

−r *rel* Initial release: the *rel*ease into which the initial delta is inserted. **−r** may be used only if the **−i** option is also used. The initial delta is inserted into release 1 if the **−r** option is not used. The level of the initial delta is always 1, and initial deltas are named 1.1 by default.

−t[*name*]
Descriptive **t**ext: The file *name* contains descriptive text for the SCCS file. The descriptive text file name *must* be supplied when creating a new SCCS file (either or both **−n** and **−i** options) and the **−t** option is used. In the case of existing SCCS files: 1) a **−t** option without a file name removes descriptive text (if any) currently in the SCCS file, and 2) a **−t** option with a file name replaces the descriptive text currently in the SCCS file with any text in the named file.

−f*flag* Set *flag*: specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **−f** options may be supplied on a single *admin* command line. *Flag*s and their values appear in the FLAGS section after this list of options.

−d*flag* Delete *flag* from an SCCS file. The **−d** option may be specified only when processing existing SCCS files. Several **−d** options may be supplied on a single *admin* command. See the FLAGS section below.

−l *list* Unlock the specified *list* of releases. See the **−f** option for a description of the **l** flag and the syntax of a *list*.

−a *login*
: Add *login* name, or numerical UNIX group ID, to the list of users who may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **−a** options may appear on a single *admin* command line. As many *logins*, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, anyone may add deltas.

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

−y [*comment*]
: The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). If the **−y** option is omitted, a default comment line is inserted in the form:
date and time created *YY/MM/DD HH:MM:SS* by *login*
The **−y** option is valid only if the **−i** and/or **−n** options are specified (that is, a new SCCS file is being created).

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

−h Check the structure of the SCCS file (see *sccsfile*(5)), and compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file.

The **−h** option inhibits writing on the file, so that it nullifies the effect of any other options supplied, and is, therefore, only meaningful when processing existing files.

−z recompute the SCCS file check-sum and store it in the first line of the SCCS file (see **−h**, above).

Using the **−z** option on a truly corrupted file may prevent future detection of the corruption.

FLAGS

The list below is a description of the *flags* which may appear as arguments to the **−f** (set flags) and **−d** (delete flags) options.

b When set, the **−b** option can be used on a *get*(1) command to create branch deltas.

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

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

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

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

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

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

<list>

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

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

n The *delta*(1) command creates a 'null' delta in each release (if any) being skipped when a delta is made in a *new* release. For example, releases 3 and 4 are skipped when making delta 5.1 after delta 2.7. These null deltas serve as 'anchor points' so that branch deltas may be created from them later. If the **n** flag is absent from the SCCS file, skipped releases will be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

q *text* *Text* is defined by the user. The *text* is substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(1).

m *module*

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

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

v [*program*]

Validity checking *program*: *delta*(1) prompts for Modification Request (*MR*) numbers as the reason for creating a delta. The optional *program* specifies the name of an *MR* number validity checking program (see *delta*(1)). If this flag is set when creating an SCCS file, the **−m** option must also be used even if its value is null.

FILES

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

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

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

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

SEE ALSO

sccs(1), delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(5).
Source Code Control System in *Programming Tools for the Sun Workstation*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar – archive and library maintainer

SYNOPSIS

ar d¦m¦p¦q¦r¦t¦x [**abcilouv**] [*posname*] *afile name* **...**

DESCRIPTION

Ar maintains groups of files combined into a single archive/library file. It is normally used to create and update library files used by the loader; it can be used, though, for any similar purpose.

One of the mandatory keys (**dmpqrtx**) must be used; it may be followed by one or more of the modifiers **abcilouv**. *Afile* is the archive file. The *names* are constituent files in the archive file.

OPTIONS

d Delete the named files from the archive file.

m Move the named files to the end of the archive.

p Display the named files in the archive.

q Quick append. Append the named files to the end of the archive file without searching the archive for duplicate names. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

r Replace the named files in the archive file.

t Display a table of contents of the archive file. If no names are given, all files in the archive are listed; if names are given, only those files are listed.

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

MODIFIERS

a Place new files after *posname* (*posname* argument must be present). Applies only with the **r** and **m** options.

b Place new files before *posname* (*posname* argument must be present). Applies only with the **r** and **m** options.

c *Ar* creates *afile* when it needs to, and displays a message to this effect. The **c** modifier suppresses this message.

i Identical to the **b** modifier.

l Local. *Ar* places its temporary files in the directory /tmp. The **l** modifier places them in the local directory.

o Old date. When files are extracted with the **x** option, **o** sets the "last modified" date to the date recorded in the archive.

u Replace only those files that have changed since they were put in the archive. Used with the **r** option.

v Verbose. Give a file-by-file description of the creation of a new archive file from the old archive and the constituent files. When used with **t,** it gives a long listing of all information about the files. When used with **p,** it precedes each file with a name.

FILES

/tmp/v* temporaries

SEE ALSO

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

BUGS

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

The 'last-modified' date of a file will not be altered by the **o** option unless the user is either the owner of the extracted file or the super-user.

NAME

as – mc68000 assembler

SYNOPSIS

as [**–d2**] [**–j**] [**–J**] [**–h**] [**–L**] [**–R**] [**–o** objfile] file

DESCRIPTION

As translates assembly code in the named *file* into executable object code in the specified *objfile.*

All undefined symbols in the assembly are treated as global.

The output of the assembly is left in the file *objfile.* If the **–o** flag is omitted, file *a.out* is used.

OPTIONS

–d2 Specifies that instruction offsets involving forward or external references, and having sizes unspecified in the assembly language are two bytes long. The default is four bytes. See also the **–j** option

–J Suppress span-dependent instruction calculations and force all branches and calls to take the most general form. This is used when assembly time must be minimized, but program size and run time are not important.

–h Suppress span-dependent instruction calculations and force all branches to be of medium length, but all calls to take the most general form. This is used when assembly time must be minimized, but program size and run time are not important. This option results in a smaller and faster program than that produced by the **–J** option, but some very large programs may not be able to use it because of the limits of the medium-length branches.

–L Save defined labels beginning with an 'L', which are normally discarded to save space in the resultant symbol table. The compilers generate such temporary labels.

–j Use short (pc-relative) branches to resolve jump's and jsr's to externals. This is for compact programs which cannot use the **-d2** flag because of large program relocation.

–R Make initialized data segments read-only by concatenating them to the text segments. This eliminates the need to run editor scripts on assembly code to make initialized data read-only and shared.

FILES

/tmp/as* default temporary file

SEE ALSO

ld(1), nm(1), adb(1), dbx(1), a.out(5)
The *"Assembler Reference Manual for the Sun Workstation"* in the Sun *Programming Tools* Manual

BUGS

Should assemble standard input with no arguments.

The Pascal compiler (*pc*(1)) qualifies a nested procedure name by chaining the names of the enclosing procedures. This sometimes results in names long enough to abort the assembler, which currently limits identifiers to 512 characters.

NAME

at — execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

At squirrels away a copy of the named *file* (standard input default) to be used as input to *sh*(1) or *csh*(1) at a specified later time. At inserts a *cd* command to the current directory at the beginning of the copy file, and follows this with assignments to all environment variables (except TERM, which is useless in this context.) When the script is run, it uses the user and group ID of the creator of the copy file.

Time is specified by from 1 to 4 digits, and may be followed by 'a', 'p', 'n' or 'm' for AM, PM, noon, or midnight (letters may be upper or lower case). One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

Day may be either a month name followed by a day number — for example, 'apr 26' — or a day of the week — 'weds', for instance. If you name a day of the week and follow this with the word 'week' — 'weds week' — invocation is moved seven days further off. Names of months and days may be recognizably truncated.

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

Standard output or error output is lost unless redirected.

EXAMPLES

Examples of legitimate commands are:

```
at 8a jan 24

at 1530 fr week
```

FILES

/usr/lib/atrun	executor (run by *cron*(8)).

in /usr/spool/at:

yy.ddd.hhhh.*	activity for year yy, day dd, hour hhhh.
lasttimedone	last *hhhh*
past	activities in progress

SEE ALSO

calendar(1), pwd(1), sleep(1), cron(8)

BUGS

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

NAME

awk — pattern scanning and processing language

SYNOPSIS

awk [**−f** *program_file*] [**−F***c*] [*program*] [*file* ...]

DESCRIPTION

Awk scans each of its input *file*s for lines that match any of a set of patterns specified in *program*. The input *file*s are read in order; the standard input is read if there are no *file*s. The filename '−' means the standard input.

The set of patterns may either appear literally on the command line as *program*, or, by using the **−f** option, the set of patterns may be in a *program_file*.

With each pattern in *program* there can be an associated action that will be performed when a line of a *file* matches the pattern. See the discussion below for the format of input lines and the *awk* language. Each line in each input *file* is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

OPTIONS

−f *program_file*
Use the contents of *program_file* as the source for the *program*.

−F *c* Use the character *c* as the field separator (FS) character. See the discussion of FS below.

LINES, STATEMENTS, AND THE AWK LANGUAGE

Input Lines

An input line is made up of fields separated by white space. The field separator can be changed by using FS — see below. Fields are denoted $1, $2, ... up to $9; $0 refers to the entire line.

Pattern-action Statements

A pattern-action statement has the form

```
pattern { action }
```

A missing { action } means copy the line to the output; a missing pattern always matches.

An action is a sequence of statements. A statement can be one of the following:

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

Format of the Awk Language

Statements are terminated by semicolons, newlines 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, providing a form of associative memory. String constants are quoted "...".

The *print* statement prints its arguments on the standard output (or on a file if *>file* 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 template (see *printf*(3S) for a description of the formatting control characters).

Built In Functions

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

Patterns

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*. 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 pattern BEGIN may be used to capture control before the first input line is read, in which case BEGIN must be the first pattern. The special pattern END may be used to capture control after the last input line is read, in which case END must be the last pattern.

Special Variable Names

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

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

or by using the **−F***c* option.

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

SEE ALSO

lex(1), sed(1)
Pattern Scanning and Processing with Awk in the Sun *Editing and Text Processing* Manual.

BUGS

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 "" to it.

Syntax errors result in the cryptic message "*awk bailing out near line 1!!.*"

NAME

basename − strip filename affixes

SYNOPSIS

basename string [suffix]

DESCRIPTION

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

EXAMPLE

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

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

SEE ALSO

sh(1)

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

bc [**−c**] [**−l**] [file **...**]

DESCRIPTION

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

```
Comments
        are enclosed in /* and */.

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

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

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

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

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

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

All function arguments are passed by value.

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

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

EXAMPLES

Define a function to compute an approximate value of the exponential function:

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

Print approximate values of the exponential function of the first ten integers:

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

OPTIONS

−l is the name of an arbitrary precision math library.

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

FILES

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

SEE ALSO

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

BUGS

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

NAME

biff — mail alarm

SYNOPSIS

biff [**y**|**n**]

DESCRIPTION

Biff informs the system whether you want to be notified when mail arrives during the current terminal session. The command:

```
biff y
```

enables notification; the command:

```
biff n
```

disables it; finally, the command:

```
biff
```

on its own tells you whether the notification is **y** or **n**. When mail notification is enabled, the header and first few lines of the message are printed on your screen whenever mail arrives. A **biff y** command is often included in the file *.login* or *.profile* to be executed at each login.

Biff operates asynchronously. For synchronous notification use the MAIL variable of *sh*(1) or the *mail* variable of *csh*(1).

SEE ALSO

csh(1), sh(1), mail(1)

NAME

/bin/mail – send or receive mail among users

SYNOPSIS

/bin/mail [**–i**] [**–p**] [**–q**] [**–f** *filename*]
/bin/mail [**–d**] [**–i**] [**–r** *name*] *person* ...

DESCRIPTION

Note: This is the old version 7 UNIX system mail program. The default *mail* command is described in mail(1), and its binary is in the directory */usr/ucb*.

/bin/mail with no argument prints a user's mail, message-by-message in last-in, first-out order. */bin/mail* accepts commands from the standard input to direct disposition messages.

When *persons* are named, */bin/mail* takes the standard input up to an end-of-file (or a line with just '**.**') and adds it to each *person's* 'mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *person* is usually a user name recognized by *login*(1).

If there is any pending mail, *login*(1) tells you there is mail when you log in. It is also possible to have the C-Shell, see *csh*(1) or the comsat daemon *biff*(1) tell you about mail that arrives while you are logged in.

OPTIONS

Printing Mail

–i continue after interrupts — an interrupt normally terminates the */bin/mail* command and leaves the mail file unchanged.

–p print messages without prompting for commands. Exit immediately upon receiving an interrupt.

–q quit immediately upon interrupt.

–f *filename*
use *filename* as if it were the mail file.

Sending Mail

–d deliver mail directly, don't route the message through *sendmail*. This option is often used by programs that send mail.

–i continue after interrupts — an interrupt normally terminates the */bin/mail* command and leaves the mail file unchanged.

–r *name*
specify a string to appear as the name of the sender.

COMMANDS

? print a command summary.

EOT (control-D)
put unexamined mail back in the mail file and quit.

!*command*
escape to the Shell to do *command*.

– go back to previous message.

+ go on to next message.

newline
go on to next message.

d delete message and go on to the next.

dq delete message and quit.

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

n go on to next message.

p print message (again).

q same as EOT.

s [*file*] ...
save the message in the named *files* ('mbox' default).

w [*file*] ...
save the message, without a header, in the named *files* ('mbox' default). Delete the message from the list and go on to the next message.

x exit without changing the mail file.

FILES

/etc/passwd	to identify sender and locate persons
/usr/spool/mail/*	incoming mail for user *
mbox	saved mail
/tmp/ma*	temp file
/usr/spool/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

biff(1), write(1), uucp(1C), uux(1C), xsend(1), sendmail(8)

BUGS

Race conditions sometimes result in a failure to remove a lock file.

Any superuser can read your mail, unless it is sent by *xsend*(1).

NAME

cal – display calendar

SYNOPSIS

cal [month] year

DESCRIPTION

Cal displays a calendar for the specified year. If a month is also specified, a calendar for that month only is displayed.

Year can be between 1 and 9999. Be aware that '*cal* 78' refers to the early Christian era, not the 20th century. Also, the year is always considered to start in January, even though this is historically naive.

Month is a number between 1 and 12.

The calendar produced is that for England and her colonies.

Try September 1752.

NAME

calendar — reminder service

SYNOPSIS

calendar [—]

DESCRIPTION

Calendar consults the file *calendar* in the current directory and displays lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates — such as 'Dec. 7,' 'december 7,' and '12/7' — are recognized; but '7 December' or '7/12' are not. If you give the month as "*" with a date — for example, "* 1" — that day in any month will do. On weekends 'tomorrow' extends through Monday.

When the optional — argument is present, *calendar* does its job for every user who has a file *calendar* in his login directory and sends him any positive results by *mail*(1). Normally this is done daily in the wee hours under control of *cron*(8).

The file *calendar* is first run through the C preprocessor, */lib/cpp*, to include any other calendar files specified with the usual "#include" syntax. Included calendars are usually shared by all users, and maintained by the system administrator.

FILES

~/calendar	
/usr/lib/calendar	to figure out today's and tomorrow's dates
/etc/passwd	
/tmp/cal*	
/lib/cpp	subprocess
/usr/bin/egrep	subprocess
/bin/sed	subprocess
/bin/mail	subprocess

SEE ALSO

at(1), cron(8), mail(1)

BUGS

Calendar's extended idea of 'tomorrow' doesn't account for holidays.

NAME

cat – concatenate and display

SYNOPSIS

cat [**–u**] [**–n**] [**–b**] [**–s**] [**–v**] [**–e**] [**–t**] [**–**] [file ...]

DESCRIPTION

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

```
% cat goodies
```

displays the contents of *goodies* on the standard output, and

```
% cat file1 file2 >file3
```

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

If no filename argument is given, or if the argument '–' is given, *cat* reads from the standard input file. If the standard input is a terminal, input is terminated by a ^D.

OPTIONS

–u makes the output completely unbuffered. If **–u** is not used, output is buffered in 1024-byte blocks, or line-buffered if standard output is a terminal.

–n precedes each line output with its line number.

–b numbers the lines, as **–n**, but omits the line numbers from blank lines.

–s substitutes a single blank line for multiple adjacent blank lines.

–v displays non-printing characters so that they are visible. Control characters print like ^X for control-x; the delete character (octal 0177) prints as ^?. Non-ASCII characters (with the high bit set) are displayed as M- (for meta) followed by the character of the low 7 bits.

–e displays non-printing characters, as **–v**, and in addition displays a '$' character at the end of each line.

–t displays non-printing characters, as **–v**, and in addition displays tab characters as '^I'.

SEE ALSO

cp(1), ex(1), more(1), pr(1), tail(1)

BUGS

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

NAME

cb — C program beautifier

SYNOPSIS

cb

DESCRIPTION

Cb places a copy of the C program from the standard input on the standard output with spacing and indentation that displays the structure of the program.

SEE ALSO

indent(1)

indent(1) is preferred.

NAME

cc — C compiler

SYNOPSIS

cc [**—c**] [**—g**] [**—o** *output*] [**—O**] [**—a**] [**—A***asmflags*] [**—B** *string*] [**—C**]
[**—D** *name*] [**—D** *name=def*] [**—E**] [**—fsingle**] [**—fsky**] [**—go**] [**—I** *dir*] [**—p**]
[**—pg**] [**—R**] [**—S**] [**—t** [**p**|**0**|**1**|**2**]] [**—U** *name*] [**—v**] [**—w**] *filename* ...

DESCRIPTION

Cc is the UNIX C compiler which translates programs written in the C programming language into executable load modules, or into relocatable binary programs for subsequent loading with the *ld*(1) linker.

In addition to the many options, *cc* accepts several types of files. Files whose names end with *.c* are taken to be C source programs; they are compiled, and each resulting object program is left in the current directory, with the same name as the source file, except that the *.c* suffix is replaced by *.o*. In the same way, files whose names end with *.s* are taken to be assembly source programs and are assembled, producing a *.o* file.

Other arguments are taken to be loader option arguments, object programs, or libraries of object programs. Unless **—c**, **—S**, or **—E** is specified, these programs and libraries, together with the results of any compilations or assemblies specified, are loaded (in the order given) to produce an executable program named *a.out*. The name *a.out* can be overridden with the loader's **—o** *name* option.

If a single C program is compiled and loaded all at once, the intermediate *.o* file is deleted.

OPTIONS

The following options are interpreted by *cc*. See *ld*(1) for load-time options.

—c Compile only — suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.

—g Produce additional symbol table information for *dbx*(1) and pass the **—lg** flag to *ld*(1).

—o *output*
Name the final output file *output*. If this option is used, the file *a.out* is left undisturbed.

—O / Use the object code optimizer to improve the generated code.

—a Insert code into the program to count how many times each basic block in the program is executed. After the program is compiled with this option, there will be a *.d* file for every *.c* file. The *.d* file accumulates execution data for the corresponding source file. The *tcov*(1) utility can then be run on the source file to generate statistics about the program. This process is performed after the C preprocessor *cpp*(1) has run but before compilation takes place. The **—a** option must be used when linking with the *cc* or *ld* command as well as the compile command.

—A*asmflags*
Pass *asmflags* to the assembler *as* in its command line. For example, **—A—J** would pass the **—J** option to tell the assembler to use short PC-relative instructions for subroutine calls.

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

—C Prevent the C preprocessor from removing comments.

—D*name*

—D*name=def*

Define *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as "1".

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

—fslngle
Use single-precision arithmetic in computations involving only **float** numbers — that is, do not convert everything to **double**, which is the default. Note that floating-point parameters are still converted to double-precision, and functions which return values still return double-precision values. Certain programs run much faster using this option, but be aware that some significance can be lost due to lower precision intermediate values.

—fsky Generate code which assumes the presence of a SKY floating-point processor board. Programs compiled with this option can only be run in systems that have a SKY board installed. Programs compiled without the **—fsky** option will use the SKY board if it is present, but won't run as fast as they would if the **—fsky** option were used. If any part of a program is compiled using the **—fsky** option, you must also use this option when linking with the *cc* command, since a different set of startup routines is used.

—go Produce additional symbol table information in an older format used by the *adb*(1) debugger and pass the **—lg** flag to *ld*(1).

—I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **—I** options, then in the */usr/include* directory.

—p Produce profiling code to count the number of times each routine is called. If loading takes place, replace the standard startup routine by one that automatically calls *monitor*(3) and use a special profiling library in lieu of the standard C library. When the program is run, the file *mon.out* is created. An execution profile can then be generated by use of *prof*(1).

—pg Produce profiling code in the manner of **—p**, but invokes a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. *gprof*(1) generates an execution profile.

—R Passed on to *as*, making initialized variables shared and read-only.

—S Compile the named C programs and leave the assembly language output on corresponding files suffixed *.s*.

—t[**p012**]
Find only the designated compiler passes in the files whose names are constructed by a **—B** option. In the absence of a **—B** option, the *string* is taken to be */usr/new/*. The letter/number combinations that can be specified for the **—t** option have the meanings:

p	*cpp* — the C preprocessor.
0	*ccom* — both phases of the C compiler, but not the optimizer.
1	Ignored in this system — this option would be for the second phase of a two-phase compiler but in the Sun system, *ccom* includes both phases.
2	*c2* — the object code optimizer.

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

—v Display a message on the standard error file indicating the argument string that invoked each phase of the compiler.

—w Suppress warning messages.

FILES

file.c	C source file
file.s	assember source file
file.o	object file
file.a	library of object files
a.out	executable output file
/tmp/ctm?	temporary
/lib/cpp	macro preprocessor
/usr/lib/bb_count	block counting preprocessor
/lib/ccom	compiler
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/usr/lib/gcrt0.o	startoff for gprof-profiling
/usr/lib/bb_link.o	basic block counting routine
/lib/fcrt0.o	SKY runtime startoff
/lib/fmcrt0.o	SKY startoff for profiling
/usr/lib/fgcrt0.o	SKY startoff for gprof-profiling
/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libc_p.a	profiling library, see *intro*(3)
/usr/include	standard directory for '#include' files
mon.out	file produced for analysis by *prof*(1)
gmon.out	file produced for analysis by *gprof*(1)

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, 1978
UNIX Programming in *Programming Tools for the Sun Workstation*
monitor(3), prof(1), gprof(1), tcov(1), adb(1), ar(1), ld(1), dbx(1), as(1), diff(1), cpp(1)

DIAGNOSTICS

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

BUGS

Options which are identical to *cc* options cannot be passed loader.

The program context given in syntax error messages is taken from the input text *after* the C preprocessor has performed substitutions. Therefore, error message involving syntax errors in or near macro references or manifest constants may be misleading.

NAME

cd — change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

Directory becomes the new working directory. The process must have execute (search) permission in *directory*. If *cd* is used without arguments, it returns you to your login directory. In *csh*(1) you may specify a list of directories in which *directory* is to be sought as a subdirectory if it is not a subdirectory of the current directory; see the description of the *cdpath* variable in *csh*(1).

SEE ALSO

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

NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

/usr/ucb/sccs/cdc **–r**SID [**–m** [mrlist]] [**–y** [comment]] file ...

DESCRIPTION

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

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

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

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

OPTIONS

All the described options apply independently to each named file:

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

–m[*mrlist*]

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

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

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

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

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

–y[*comment*]

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

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

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

EXAMPLES

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

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

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

does the same thing.

WARNINGS

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

FILES

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

SEE ALSO

admin(1), comb(1), delta(1), get(1), help(1), prs(1), sccs(1), sccsdiff(1), sccsfile(5), val(1), what(1).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

checknews – check if user has news on the USENET news network

SYNOPSIS

checknews [**ynqevv**] [*readnews options*]

DESCRIPTION

checknews reports to the user whether or not there is news.

OPTIONS

y Reports 'There is news' if the user has news to read. **y** is the default if no options are specified.

n Reports 'No news' if there isn't any news to read.

q Makes *checknews* quiet. Instead of displaying a message, the exit status indicates news. A status of 0 means no news, 1 means there is news.

v alters the **y** message to show the name of the first newsgroup containing unread news. Doubling **v** (that is, **vv**) displays a report of *any* claim of new news, and is useful if **checknews** and **readnews** disagree on whether there is news.

e Executes *readnews*(1) if there is news.

FILES

~/.newsrc	Options and list of previously read articles
/usr/lib/news/active	Active newsgroups

SEE ALSO

inews(1), postnews(1), readnews(1)
Network News User's Guide in the *Beginner's Guide to the Sun Workstation.*

NAME

checknr – check nroff/troff files

SYNOPSIS

checknr [**–s**] [**–f**] [**–a** *.x1.y1.x2.y2 . . . .xn.yn*] [**–c** *.x1.x2.x3 . . . .xn*] [*filename . . .*]

DESCRIPTION

Checknr checks a list of *nroff*(1) or *troff*(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. _If no files are specified, *checknr* checks the standard input. Delimiters checked are:

(1) Font changes using \f*x* ... \fP.

(2) Size changes using \s*x* ... \s0.

(3) Macros that come in open ... close forms, for example, the .TS and .TE macros which must always come in pairs.

Checknr knows about the *ms*(7) and *me*(7) macro packages.

Checknr is intended to be used on documents that are prepared with *checknr* in mind. It expects a certain document writing style for \f and \s commands, in that each \fx must be terminated with \fP and each \sx must be terminated with \s0. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from *checknr.* Since it is probably better to use the \fP and \s0 forms anyway, you should think of this as a contribution to your document preparation style.

OPTIONS

–s Ignore \s size changes.

–f Ignore \f font changes.

–a Add pairs of macros to the list. The pairs of macros are assumed to be those (such as .DS and .DE) that should be checked for balance. The **–a** option must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair .BS and .ES, use **–a**.BS.ES

–c define commands which *checknr* would otherwise complain about as undefined.

SEE ALSO

nroff(1), troff(1), ms(7), me(7), checkeq(1)

BUGS

There is no way to define a 1 character macro name using **–a**

NAME

chgrp − change group

SYNOPSIS

chgrp [**−f**] group file ...

DESCRIPTION

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

The user invoking *chgrp* must belong to the specified group and be the owner of the file, or be the super-user.

No errors are reported when the **−f** (force) option is given.

FILES

/etc/group

SEE ALSO

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

NAME

chmod – change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

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

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

A symbolic *mode* has the form:

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

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for all, or **ugo.** If *who* is omitted, the default is **a** but the setting of the file creation mask (see umask(2)) is taken into account.

Op can be **+** to add *permission* to the file's mode, – to take away *permission* and = to assign *permission* absolutely (all other bits for that category, owner, group, or others, will be reset).

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

EXAMPLES

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

```
chmod o–w file
chmod +x file
```

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

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

SEE ALSO

ls(1), chmod(2), stat(2), umask(2), chown(8)

NAME

chsh − change default login shell

SYNOPSIS

chsh username [shell]

DESCRIPTION

Chsh changes the login shell field of the user's password file entry. If no ***shell*** is specified, the shell reverts to the default login shell */bin/sh.* To specify a shell other than */bin/csh,* you must be the super-user.

EXAMPLES

```
angel% chsh bill /bin/csh
```

SEE ALSO

csh(1), passwd(1), passwd(5)

NAME

clear — clear 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

clear_colormap – clear the color map

SYNOPSIS

clear_colormap

DESCRIPTION

Clear_colormap clears your hardware colormap.

NAME

clocktool – display the time in a window

SYNOPSIS

clocktool [**–s**] [**–t**] [**–r**] [**–d mdyaw**] [**–f**]

DESCRIPTION

Clocktool displays the current time in a window. While it is open it prints the date and time in its window; when it is closed, its icon is a clock face which keeps time.

OPTIONS

–r causes *clocktool* to use a square face with roman numerals in the iconic state. This replaces the default round clock face.

–d display date information in a small area just below the clock face. The date information to be displayed may include:

m	the month,
d	the day of the month (1-31),
y	the year,
a	the string **AM** or **PM**, as appropriate,
w	the day of the week (Sun–Sat).

There is only room for 3 of these, but any 3 may be displayed in any sequence.

–s start *clocktool* with the seconds turned on. By default, the clock starts with seconds turned off, and updates every minute. With seconds turned on, it updates every second, and, if iconic, displays a second hand.

–t Ignore the real time, and instead run in a loop continuously incrementing the time by one minute and displaying it.

–f Display the date and day of week on the clock face.

Clocktool also accepts all of the generic tool arguments discussed in *suntools*(1).

Clocktool's window is not a terminal subwindow; it provides the Tool Manager menu if you depress the menu button in it. However, it listens for keyboard input, toggling its state on two letters (case does not matter):

s toggles the display of seconds.

t toggles the 'test' mode.

You can only modify these states while the clock is open; when it is iconic, it responds with the standard Tool Manager functions to mouse and keyboard input.

SEE ALSO

suntools(1).

FILES

/usr/lib/fonts/fixedwidthfonts/sail.r.6

BUGS

There should be a way to change the date information on the fly.

The date display doesn't go well with the round clock face.

NAME

cmp – compare two files

SYNOPSIS

cmp [**–l**] [**–s**] file1 file2

DESCRIPTION

Cmp compares *file1* and *file2*. If *file1* is '–', *cmp* reads from the standard input. Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

OPTIONS

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

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

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

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

NAME

col − filter reverse paper motions

SYNOPSIS

col [−bfx]

DESCRIPTION

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

The control characters SO (ASCII code 017), and SI (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by one of 7, 8, 9, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

OPTIONS

−f Fine: 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. The **−f** option suppresses this treatment — in this case the output from *col* may contain forward half line feeds (ESC-9), but will still never contain either kind of reverse line motion.

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

−x Do not convert white space to tabs to shorten printing time.

SEE ALSO

troff(1), tbl(1)

BUGS

Can't back up more than 128 lines.
No more than 800 characters, including backspaces, on a line.

NAME

colcrt – filter nroff output for CRT previewing

SYNOPSIS

colcrt [–] [**–2**] [file ...]

DESCRIPTION

Colcrt provides virtual half-line and reverse line feed sequences for terminals without such capability, and on which overstriking is destructive. Half-line characters and underlining (changed to dashing '–') are placed on new lines in between the normal output lines.

OPTIONS

– Suppress all underlining — especially useful for previewing *allboxed* tables from *tbl*(1).

–2 Print all half-lines, effectively double spacing the output. Normally, a minimal space output format is used which suppresses empty lines. *Colcrt* never suppresses two consecutive empty lines, however. The **–2** option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

EXAMPLE

A typical use of *colcrt* would be

```
tbl exum2.n ¦ nroff –ms ¦ colcrt – ¦ more
```

SEE ALSO

nroff(1), troff(1), col(1), more(1), ul(1)

BUGS

Should fold underlines onto blanks even with the '–' option so that a true underline character would show; if we did this, however, *colcrt* wouldn't get rid of *cu'd* underlining completely.

Can't back up more than 102 lines.

General overstriking is lost; as a special case '¦' overstruck with '–' or underline becomes '+'.

Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.

NAME

colrm – remove columns from a file

SYNOPSIS

colrm [startcol [endcol]]

DESCRIPTION

Colrm removes selected columns from a text file. The text is is taken from standard input and copied to the standard output with the specified columns removed.

If only *startcol* is specified, the columns of each line are removed starting with *startcol* and extending to the end of the line. If both *startcol* and *endcol* are specified, all columns between *startcol* and *endcol*, inclusive, are removed.

Column numbering starts with column 1.

SEE ALSO

expand(1)

NAME

comb – combine SCCS deltas

SYNOPSIS

/usr/sccs/comb [**−o**] [**−s**] [**−p** *SID*] [**−c** *list*] *filename* ...

DESCRIPTION

Comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

OPTIONS

Options are explained as though only one named file is to be processed, but the effects of any option apply independently to each named file.

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

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

−o For each **get −e** generated, the reconstructed file is accessed at the release of the delta to be created. In the absence of the **−o** option, the reconstructed file is accessed at the most recent ancestor. Use of the **−o** option may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

−s Generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original – combined) / original

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

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

FILES

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

SEE ALSO

sccs(1), admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System in *Programming Tools for the Sun Workstation*.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

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

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [– [**123**]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2,* which should be ordered in ASCII collating sequence, and produces a three column output: lines only in *file1;* lines only in *file2;* and lines in both files. The filename '—' means the standard input.

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

SEE ALSO

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

NAME

compact, uncompact, ccat − compress and uncompress files, and cat them

SYNOPSIS

compact [filename **...**]
uncompact [filename **...**]
ccat [filename **...**]

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted to the standard output. *Compact* operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, *compact* and *uncompact* can operate as filters. In particular:

```
        ... | compact | uncompact | ...
```

operates as a (very slow) no-op.

When an argument *file* is given, it is compacted and the resulting file is placed in *file.C; file* is removed. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

Uncompact restores the original file from a file called *file.C* which was compressed by *compact.* If no file names are given, the standard input is uncompacted to the standard output.

Ccat cats the original file from a file compressed by *compact,* without uncompressing the file.

FILES

*.C	compacted file created by compact, removed by uncompact

SEE ALSO

Gallager, Robert G., 'Variations on a Theme of Huffman', *I.E.E.E. Transactions on Information Theory,* vol. IT-24, no. 6, November 1978, pp. 668 - 674.

NAME

cp − copy files

SYNOPSIS

cp [**−i**] [**−r**] file1 file2

cp [**−i**] [**−r**] file **...** directory

DESCRIPTION

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

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

Cp refuses to copy a file onto itself.

OPTIONS

−i Interactive: prompt the user with the name of the file whenever the copy would overwrite an old file. Answering with 'y' means that *cp* should go ahead and copy the file. Any other answer will prevent *cp* from overwriting the file.

−r Recursive: if any of the source files are directories, *cp* copies each subtree rooted at that name; in this case the destination must be a directory.

EXAMPLES

To make a backup copy of *goodies*:

```
% cp goodies old.goodies
```

To copy an entire directory hierarchy:

```
% cp −r /usr/wendy/src  /usr/wendy/backup
```

However, **BEWARE** of a recursive copy like this one:

```
% cp −r /usr/wendy/src  /usr/wendy/src/backup
```

which keeps copying files until it fills the entire file system.

SEE ALSO

cat(1), pr(1), mv(1), rcp(1C)

BUGS

There should be an option to copy timestamps to the new files — for instance, when copying a whole hierarchy from one file system to another file system, or when making a backup copy.

NAME

cpio — copy file archives in and out

SYNOPSIS

cpio −o [**acBvs**]

cpio −i [**Bcdmrtuv6s**] [patterns]

cpio −p [**adlmruvs**] directory

DESCRIPTION

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

Cpio −i (copy in) reads the standard input (which is assumed to be the product of a previous **Cpio −o** command), to get a list of files selected by zero or more *patterns* as defined in the name-generating notation of *sh*(1) or *csh*(1). In *patterns*, the meta-characters ?, *, and [...] match the slash (/) character. The default for *patterns* is * (select all files).

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

OPTIONS

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

B Input/output is to be blocked at 5120 bytes to the record. This does not apply to the *pass* option. This option is only meaningful with data directed to or from **/dev/rmt?**

d *Directories* should be created as needed.

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

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

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

u Copy *unconditionally*. Normally, an older file will not replace a newer file with the same name.

v *Verbose* option. A list of filenames is displayed. When used with the **t** option, the table of contents looks like the output of an **ls −l** command (see *ls*(1)).

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

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

6 Process an old (version 6 UNIX system) file. This is only useful with **−i** (copy in).

s Swaps pairs of data bytes. Note that the **s** option cannot be used with the **c** option.

EXAMPLES

To copy the contents of a directory into an archive:

```
% ls | cpio −o > /dev/mt0
```

To duplicate the *olddir* directory hierarchy in the *newdir* directory:

```
% cd olddir
% find . −print | cpio −pdl newdir
```

Some forms of *cpio* tapes from other sites have the bytes swapped in the file. The **s** option doesn't help since it only swaps the data bytes and not the header. To overcome this problem, use *dd* with the **conv=swab** option to swap *all* pairs of bytes (including the header), then pipe the output of *dd* through *cpio* with the **s** option to swap the data bytes back again:

```
% dd if=whatever the file is conv=swab | cpio −is
```

SEE ALSO

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

BUGS

Pathnames are restricted to 128 characters. If there are too many unique linked files, *cpio* runs out of memory to keep track of them and linking information is lost thereafter. Only the super-user can copy special files.

NAME

cpp – the C language preprocessor

SYNOPSIS

/lib/cpp [**–P –C –U***name* **–D***name* **–D***name=def* **–I***dir*] [*ifile* [*ofile*]]

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command (*cpp* may optionally be invoked as the first pass of a FORTRAN 77 or Pascal compilation — see *f77*(1) or *pc*(1)). Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the compiler. The preferred way to invoke *cpp* is through the *cc*(1) command. See *m4*(1) for a general macro processor.

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

OPTIONS

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

–C Pass all comments (except those which appear on *cpp* directive lines) through the preprocessor. By default, *cpp* strips C-style comments.

–U*name*

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

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, vax, mc68000
UNIX System variant:	RES, RT, sun

–D*name*

Define *name* as 1 (one). This is the same as if a **–D***name=1* option had appeared on the *cpp* command line, or as if a **#define name 1** line had appeared in the source file that *cpp* is processing.

–D*name=def*

Define *name* as if by a **#define** directive. This is the same as if a **#define name def** line had appeared in the source file that *cpp* is processing.

–I*dir* Change the algorithm for searching for **#include** files whose names do not begin with **/** to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in **" "** will be searched for first in the directory of the current source file, then in directories named in **–I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in **<>**, the directory of the *ifile* argument is not searched. See the section entitled DETAILS below, for exact details of the search order.

–R Allow recursive macros.

DIRECTIVES

All *cpp* directives start with lines begun by **#**. White space (blanks or tabs) can appear after the **#**. The directives are:

#define *name token-string*

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

#define *name* (*arg*, ..., *arg*) *token-string*

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

#undef *name*

Forget the definition of *name* (if any) from now on.

#include *"filename"*

#include *<filename>*

Include at this point the contents of *filename* (which is then run through *cpp*). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See DETAILS below.

#line *integer-constant "filename"*

Generate line control information for the next pass of the C compiler. *Integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file where it comes from. If *"filename"* is not given, the current file name is unchanged.

#endif *commentary*

Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. The *commentary* can be used to associate the **#endif** with its opening **#if**.

#ifdef *name*

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

#ifndef *name*

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

#if *constant-expression*

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

#else *commentary*

Reverses for the following lines the notion of the test directive currently in effect. So if lines previous to this directive are ignored, the following lines will appear in the output, and vice versa. The *commentary* can be used to associate the **#else** with its opening **#if**.

The test directives and corresponding **#else** directives can be nested.

DETAILS

Directory search order for #include files is:

1. the directory of the file which contains the #include request (that is, #include is relative to the file being scanned when the request is made)
2. the directories specified by –I options, in left-to-right order.
3. the standard directory(s) (*/usr/include* for the Sun system).

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

A newline terminates a character constant or quoted string.

An escaped newline (that is, a backslash immediately followed by a newline) may be used in the body of a '#define' statement to continue the definition onto the next line. The escaped newline is not included in the macro body.

Comments are removed (unless the –C option is used on the command line). Comments are also ignored, except that a comment terminates a token.

Macro formal parameters are recognized in '#define' bodies even inside character constants and quoted strings. The output from:

```
#define foo(a) '\a'
foo(bar)
```

is the seven characters '\bar' (*space*, single-quote, escape character, b, a, r, single-quote). Macro names are not recognized inside character constants or quoted strings during the regular scan. Thus:

```
#define foo bar
printf("foo");
```

does not expand 'foo' in the second line, because it is inside a quoted string which is not part of a '#define' macro definition.

Macros are not expanded while processing a '#define' or '#undef'. Thus:

```
#define foo bletch
#define bar foo
#undef foo
bar
```

produces 'foo'. The token appearing immediately after a '#ifdef' or '#ifndef' is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces ' goodbye' (and warns about the redefinition of 'greeting').

Output consists of the input file, with modifications, plus lines of the form: *#lineno "filename"* –indicating the original source line number and filename of the following output line.

FILES

/usr/include	standard directory for #include files

SEE ALSO

cc(1), m4(1), f77(1), pc(1).

DIAGNOSTICS

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

NOTES

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

NAME

crypt – encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

Crypt encrypts and decrypts the contents of a file. *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:

```
        tutorial% crypt key <clear.file >encrypted.file
        tutorial% crypt key <encrypted.file ¦ pr
```

will print the contents of *clear.file*.

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

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; 'sneak paths' by which keys or 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, that is, to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

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

FILES

/dev/tty for typed key

SEE ALSO

ed(1), ex(1), makekey(8) and vi(1)

RESTRICTIONS

This program is not available on software shipped outside the U.S.

NAME

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

SYNOPSIS

csh [**−cefinstvVxX**] [arg ...]

DESCRIPTION

Csh is a command language interpreter which may be used instead of *sh*(1), typically to take advantage of its history mechanism (see ***History Substitutions***) and its job control facilities (see ***Jobs***).

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

In the normal case, the shell then begins reading commands from the terminal, prompting with ‘% ’. Processing of arguments and the use of the shell to process files containing command scripts is 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 in ‘&&’, ‘||’, ‘<<’, or ‘>>’ these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with ‘\’. A newline preceded by a ‘\’ is equivalent to a blank.

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

When the shell's input is not a terminal, the character ‘#’ introduces a comment which continues to the end of the input line. The ‘#’ character does not introduce a comment 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 ‘|’ (vertical bar) characters forms a ***pipeline***. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ‘;’, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an ‘&’.

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

Jobs

The shell associates a *job* with each pipeline. It keeps a table of current jobs, which you can display with 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 this job is job number 1 and has one (top-level) process, whose process id is 1234.

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

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

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

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

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

Substitutions

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

History substitutions

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

Commands input from the terminal which consist of one or more words are saved on the history list, the size of which is controlled by the *history* variable; the previous command is always retained, regardless of its value. The history substitutions reintroduce sequences of words from these saved commands into the input stream. Commands are numbered sequentially from 1.

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

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

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

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

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

#	the entire command line typed so far
0	first (command) word
n	n'th argument
^	first argument, that is, '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 *r* for *l*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, for example, ‘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, for example, ‘!$’. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus ‘!?foo?^ !$’ gives the first and last arguments from the command matching ‘?foo?’.

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

Quotations with ´ and "

The quotation of strings by ‘´’ and ‘"’ can be used to prevent all or some of the remaining substitutions. Strings enclosed in ‘´’ are prevented any further interpretation. Strings enclosed in ‘"’ are yet variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command 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, 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, 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’, ‘lookup bill’ would map to ‘grep bill /etc/passwd’.

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

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

Variable substitution

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

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 expands to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

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

$name
${name}

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

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

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

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

$0
Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

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

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

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

The current implementation allows only one ':' modifier on each '$' expansion.

The following substitutions may not be modified with ':' modifiers.

$?name
${?name}
Substitutes the string '1' if name is set, '0' if it is not.

$?0
Substitutes '1' if the current input filename is known, '0' if it is not.

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

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

Command and filename substitution

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

Command substitution

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

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

Filename substitution

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

In matching filenames, the character '.' at the beginning of a filename or immediately following a '/', as well as the character '/' must be matched explicitly. The character '*' matches any string of characters, including the null string. The character '?' matches any single character. The sequence '[...]' matches any one of the characters enclosed. Within '[...]', a pair of characters separated by '—' matches any character lexically between the two.

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

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

Input/output

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

< name

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

<< word

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

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

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

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

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

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

Uses file *name* as standard output like '>' but places output at the end of the file. If the variable *noclobber* is set, 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, the process blocks and the user is 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, for example, '*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

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

@ grab = $grab+2

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, that is, '0'. Command executions succeed, returning true, that is, '1', if the command exits with status 0, otherwise they fail, returning false, that is, '0'. If more detailed status information is required, 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, it is executed in a subshell.

alias
alias name
alias name wordlist

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*. Putting single quote signs (apostrophes) around *wordlist* and placing a \ character in front of any '!' signs in *wordlist* often solves any obscure problems with aliases.

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.

bg
bg %job ...

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

break

Resumes execution 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

Breaks from a *switch*, resuming after the *endsw*.

case label:

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

cd
cd name
chdir
chdir name

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

is tried to see if it is a directory.

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

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

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

echo wordlist
echo −n wordlist
The specified words are written to the shell's standard output, separated by spaces, and terminated with a newline unless the **−n** option is specified.

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

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

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

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

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

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

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

hashstat

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

history
history *n*
history −r *n*
history −h *n*

Displays the history event list; if *n* is given, display only the *n* most recent events. The **−r** option reverses the order of printout to be most recent first rather than oldest first. The **−h** option means display the history list 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, 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, the commands to the first *else* are executed; else if *expr2* is true, the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else.)*

jobs
jobs −l

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

kill %job
kill −sig %job ...
kill pid
kill −sig pid ...
kill −l

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

limit
limit *resource*
limit *resource maximum-use*

Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource.* If no *maximum-use* is given, the current limit is printed; if no *resource* is given, all limitations are given.

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

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

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

login

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

logout

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

nice
nice +number
nice command
nice +number command

The first form sets the *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 placed on commands in simple *if* statements apply.

nohup
nohup command

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

notify
notify %job ...

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

onintr
onintr −
onintr label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts (terminates shell scripts and returns to the terminal command input level). The second form 'onintr −' ignores all interrupts. The final form makes the shell execute a 'goto label' when an interrupt is received or a child process terminates because it was interrupted.

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

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

pushd
pushd name
pushd +n
With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (as in *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
Recompute the internal hash table of the contents of the directories in the *path* variable. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

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

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

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

setenv name value
Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variables, USER, TERM, and PATH, are automatically imported to and exported from the *csh* variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

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

source name
source −h name

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

stop
stop %job ...

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

suspend

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

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

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

time
time command

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

umask
umask value

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

unalias pattern

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

unhash

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

unlimit *resource*
unlimit

Removes the limitation on *resource*. If no *resource* is specified, all *resource* limitations are removed.

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, 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 source of input is a terminal.

%job

Brings the specified job into the foreground.

%job **&**

Continues the specified job in the background.

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

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

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

Special postfix '++' and '−−' operators increment and decrement *name* respectively, that is, '@ i++'.

Note that there must be a space after the '@' sign.

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 similarly handled; it is only necessary to worry about its setting 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. It could be set once in the *.login* except that commands through the network would not see the definition.

argv Set to the arguments to the shell. Positional parameters are substituted from this variable: '$1' is replaced by '$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. Echoes each command and its arguments just before execution. 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. If you use an overly-large value for *history*, the shell may run 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 signals from terminals. This prevents shells from being accidentally killed by control-D's.

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

If the first word of the value of *mail* is numeric it specifies a different mail checking interval, in seconds, than the default, which is 5 minutes.

If multiple mail files are specified, 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, that is, '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, 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 is 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. Overly-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. The **time** variable can be supplied with one or two values, such as 'set time=3' or 'set time=(3 "%E %P%")'. The first value is a number — *n* for instance. The shell displays a resource-usage summary for any command running for more than *n* CPU seconds. The second value is optional and is a character string which determines which resources the user wishes displayed. The character string can be any string of text with embedded control key-letters in it. A control key-letter is a percent sign (%) followed by a single *upper-case* letter. To print a percent sign, use two percent signs in a row. Unrecognized key-letters are simply printed. The control key-letters are:

D	Average amount of unshared data space used in Kilobytes.
E	Elapsed (wallclock) time for the command.
F	Page faults.
I	Number of block input operations.
K	Average amount of unshared stack space used in Kilobytes.
M	Maximum real memory used during execution of the process.
O	Number of block output operations.
P	Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time.
S	Number of seconds of CPU time consumed by the kernel on behalf of the user's process.
U	Number of seconds of CPU time devoted to the user's process.
W	Number of swaps.
X	Average amount of shared memory used in Kilobytes.

The default resource-usage summary is a line of the form:

*uuu.u*u *sss.s*s *ee:ee* *pp*% *xxx*+*ddd*k *iii*+*ooo*io *mmm*pf+*ww*w

where *uuu.u* is the user time (U), *sss.s* is the system time (S), *ee:ee* is the elapsed time (E), *pp* is the percentage of CPU time versus elapsed time (P), *xxx* is the average shared memory in Kilobytes (X), *ddd* is the average unshared data space in Kilobytes (D), *iii* and *ooo* are the number of block input and output operations respectively (I and O), *mmm* is the number of page faults (F), and *ww* is the number of swaps (W).

verbose Set by the **—v** command line option, displays the words of each command after history substitution.

Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via *execve*(2). Each word in the variable ***path*** names a directory from which the shell will attempt to execute the command. If it is given neither a **—c** nor a **—t** option, the shell hashes 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, it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for ***shell*** then the words of the alias are prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (for example, '$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 invokers home directory.

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

—n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

—s Command input is taken from the standard input.

—t A single line of input is read and executed. A '\' may be used to escape the newline at the end of this line and continue onto another line.

—v Sets the *verbose* variable, so that command input is echoed after history substitution.

—x Sets the *echo* variable, so that commands are echoed immediately before execution.

—V Sets the *verbose* variable even before '.cshrc' is executed.

—X Is to **—x** as **—V** is to **—v**.

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

Signal handling

The shell normally ignores *quit* signals. Jobs running detached (either by '&' or the *bg* or **%... &** commands) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shell's 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'.

FILES

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

LIMITATIONS

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

SEE ALSO

sh(1), access(2), execve(2), fork(2), killpg(2), pipe(2), umask(2), getrlimit(2), setrlimit(2), sigvec(2), wait(2), tty(4), a.out(5), environ(5), *Using the C-Shell* in the *Beginner's Guide to the Sun Workstation*

BUGS

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

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

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

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

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

It should be possible to use the ':' modifiers on the output of command substitutions. There are two problems with ':' modifier usage on '$' substitutions: not all of the ':' modifiers are available; only one modifier per command substitution is allowed.

Quoting conventions are contradictory and confusing.

Symbolic links fool the shell. In particular, *dirs* and 'cd ..' don't work properly once you've crossed through a symbolic link.

set path should remove duplicate pathnames from the pathname list. These often occur because a shell script or a *.cshrc* file does something like **set path=(/usr/local /usr/hosts $path)** to ensure that the named directories are in the pathname list.

There is no way to direct error output to one place and standard output to another place, except by invoking a sub-shell as follows:

tutorial% (*command* > *outfile*) >& *errorfile*

NAME

ctags — create a tags file

SYNOPSIS

ctags [**−BFatuwvx**] file **. . .**

DESCRIPTION

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

Files whose name ends in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C definitions.

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

OPTIONS

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

−F use forward searching patterns (/.../) (default).

−B use backward searching patterns (?...?).

−a append to tags file.

−t create tags for typedefs.

−w suppress warning diagnostics.

−u update the specified files in tags, that is, all references to them are deleted, and the new values are appended to the file. Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.

FILES

tags	output tags file

SEE ALSO

ex(1), vi(1)

BUGS

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

Does not know about #ifdefs.

NAME

date — display or set the date

SYNOPSIS

date [**−u**] [yymmddhhmm [**.**ss]]

DESCRIPTION

Date displays the current date and time when used without an argument.

Only the super-user may **set** the date. *yy* is the last two digits of the year; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *.ss* (optional) specifies seconds. The year, month and day may be omitted; the current values are supplied as defaults.

OPTIONS

−u Display the date in GMT (universal time). The system operates in GMT; *date* normally takes care of the conversion to and from local standard and daylight time. **−u** may also be used to set GMT time.

EXAMPLE

```
date 10080045
```

sets the date to Oct 8, 12:45 AM.

FILES

/usr/adm/wtmp	to record time-setting

SEE ALSO

utmp(5)

DIAGNOSTICS

‘Failed to set date: Not owner’ if you try to change the date but are not the super-user.

NAME

dbx – debugger

SYNOPSIS

dbx [**–r**] [**–i**] [**–I** *dir*] [*objfile* [*corefile*]]

DESCRIPTION

Dbx is a tool for source-level debugging and execution of programs. It accepts the same commands as *dbxtool* (1), but provides a user interface which does not use the window system.

Objfile is an object file produced by *cc*(1), *f77*(1) and *pc*(1) (or a combination of them) with the appropriate flag (**–g**) specified to produce symbol information in the object file. IMPORTANT: every stage of the compilation process, including the *ld* phase, must include the **–g** option.

If no *objfile* is specified, one uses the *debug* command to specify the program to be debugged. The object file contains a symbol table which includes the names of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named *core* exists in the current directory or a *corefile* is specified, *dbx* can be used to examine the state of the program when it faulted.

Debugger commands in the file *.dbxinit* are executed immediately after the symbolic information is read, if that file exists in the current directory, or in the user's home directory if *.dbxinit* doesn't exist in the current directory.

OPTIONS

–r Execute *objfile* immediately. Parameters follow the object file name (redirection is handled properly). If the program terminates successfully, *dbx* exits. Otherwise, *dbx* reports the reason for termination and waits for user response. *Dbx* reads from */dev/tty* when **–r** is specified and standard input is not a terminal.

–i Force *dbx* to act as though standard input is a terminal.

–I *dir* Add *dir* to the list of directories that are searched when looking for a source file. Normally *dbx* looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the **use** command.

DBX AND DBXTOOL COMMON USER INTERFACE

The following material concerning the user interface applies to both *dbx* and *dbxtool*. Note that commands, **toolenv**, **button**, and **unbutton** affect only *dbxtool*; they are described in the *dbxtool* (1) manual entry. The commands described below are divided into the following categories:

Execution and Tracing
Run the program being debugged, trace its execution, and stop at selected places.

Displaying and Naming Data
Display data and locate variables in the debugged program.

Accessing Source Files
Provide operations (such as editing) on the actual source text of the program.

Machine Level Commands
Provide access to memory locations and machine registers.

Miscellaneous
Miscellaneous commands such as **help**, and call a shell.

The most useful basic commands to know about are **run** to run the program being debugged, **where** to obtain a stack trace with line numbers, **print** for displaying variables, and **stop** for setting breakpoints.

File Names

File names within *dbx* may include shell metacharacters. The shell used for pattern matching is determined by the SHELL environment variable.

Expressions

Dbx Expressions are combinations of variables, constants, procedure calls, and operators. Hexadecimal constants must be preceded by a '0x' and octal constants by a '0'. Character constants must be enclosed in single quotes. Expressions cannot involve strings, structures, or arrays, although elements of structures or arrays may be used.

Dbx recognizes these operators:

+ − * / *div* %
    add, subtract, multiply, divide, integer divide and remainder

<< >> & | ~
    left shift, right shift, bitwise and, bitwise or, and bitwise complement

& *
    address of operator and contents of operator

< > <= >= == != !
    less than, greater than, less than or equal, greater than or equal, equal to, not equal to, and not

&& ||
    logical and, and logical or

sizeof (cast)
    sizeof variable or type, and type cast

The field reference operator ('.') can be used with pointers to records, as well as records, making the C operator '−>' unnecessary (although it is supported).

Precedence and associativity of operators are the same as in C. Parentheses can be used for grouping.

Of course, if the program being debugged is not active and there is no *core* file, one may only use expressions containing constants. Procedure calls require an active child process.

Dbx and FORTRAN

Note the following when using *dbx* with FORTRAN programs:

1. Array elements must be referenced with square brackets '[' and ']' rather than parentheses, e.g., **print whatsit[3]** instead of **print whatsit(3)**.
2. The main routine is referenced as MAIN (as distinguished from 'main'). All other names in the source file which have upper case letters in them will be lower case in *dbx*, unless the **−U** option of *f77* is used to compile the program.
3. When referring to the value of a logical type in an expression, use the value 0 or 1 rather than *false* or *true*, respectively.

Dbx Scope Rules

Dbx has two external variables which it uses to resolve scope conflicts. These are called 'file' and 'func' — see Section 3 below, *Accessing Source Files*. The values of 'file' and 'func' change automatically as files and routines are entered and exited during execution of the user program. 'File' and 'func' can also be changed by the user. Changing 'func' also changes the value of 'file'; however, changing 'file' does not change 'func'.

'Func' is used for name resolution, as in the command: **print grab** where grab may be defined in two different routines. The search order is:

1. Search for 'grab' in the routine named by 'func',
2. If 'grab' doesn't exist in the routine named by 'func', the file containing the routine named by 'func' is searched,
3. finally the next outer level — the whole program in the case of C and FORTRAN — is searched for 'grab'.

Clearly, if 'grab' is local to a different routine than the one named by 'func', or is a static variable in a different file than is the routine named by 'func', it won't be found. Note, however, that **print a.grab** is allowed, as long as routine 'a' has been entered but not exited yet. Note that the file that routine 'a' is in might have to be specified when the file name (minus its suffix) is the same as a routine name. For example, if routine 'a' is found in module 'a.c', then **print a.grab** would be not enough — one would have to use **print a.a.grab**. If in doubt as to how to specify a name, use the **whereis** command, as in **whereis grab** to display the full qualifications of all instances of the specified name (**grab** in this case).

'File' is used to:

1. resolve conflicts when setting 'func' — for example, when a C program has two static routines with the same name,
2. determine which file to use for commands which take only a source line number — for example, **stop at 55**, and:
3. determine which file to use for commands such as the 'edit' command which has optional arguments or no arguments at all.

When *dbx* begins execution, the value of 'file' is the first source file in the compiler/load list and the value of 'func' is the module for this same file (that is, without the suffix '.c', '.f' or '.p'). This causes name resolution to take place on a global level, that is, names are resolved such that the most global element is found. After execution begins, however, 'func' always has a value corresponding to a routine name.

Note that changing 'func' doesn't affect the place where *dbx* continues execution when the program is restarted.

1. Execution and Tracing Commands

When the process being debugged is executing, it may be interrupted by typing the interrupt character. The process is stopped and *dbx* becomes ready to accept commands. The process can be continued again with the **cont** command.

run [*args*] [< *filename*] [> *filename*]

Start executing *objfile*, passing *args* as command line arguments; < or > can be used to redirect input or output in the usual manner. Otherwise, all characters in *args* are passed through unchanged. If no arguments are specified, the argument list in the last previous **run** command (if any) is used. If *objfile* has been written since the last time the symbolic information was read in, *dbx* reads in the new information.

cont [**at** *source-line-number*] [*number*]

Continue execution from where it stopped, or, if the clause '**at** *source-line-number*' is given, from that line number. The *number* causes execution to continue as if that signal had occurred.

Source-line-number is evaluated relative to the current 'file' and must be within the current procedure/function. Execution cannot be continued if the process has 'finished', i.e., called the standard procedure '_exit'. *Dbx* captures control when the process attempts to exit, thereby letting the user examine the program state.

trace [**in** *procedure/function*] [**if** *condition*]

trace *source-line-number* [**if** *condition*]
trace *procedure/function* [**if** *condition*]
trace *expression* **at** *source-line-number* [**if** *condition*]
trace *variable* [**in** *procedure/function*] [**if** *condition*]

Display tracing information when the program is executed. A number is associated with the command and can be used to turn the tracing off (see the **delete** command).

If no argument is specified, each source line is displayed before it is executed. Execution is substantially slower during this form of tracing.

The clause '**in** *procedure/function*' restricts tracing information to be displayed only while executing inside the given procedure or function. Note that the '*procedure/function*' traced must be visible in the scope in which the trace command is issued — see the '*func*' command.

Condition is a Boolean expression and is evaluated prior to displaying the tracing information; the information is displayed only if *condition* is true.

The first argument describes what is to be traced. The effects of different kinds of arguments are described below:

Source Line Number
Display the line immediately prior to executing it. Source line numbers in a file other than the current one must be preceded by the name of the file in quotes and a colon, for example, "mumble.p":17.

Procedure or Function Name
Every time the procedure or function is called, display information telling what routine called it, from what source line it was called, and what parameters were passed to it. In addition, its return is noted, and if it is a function, the return value is also displayed.

Expression
If the argument is an *expression* with an **at** clause, the value of the expression is displayed whenever the identified source line is reached.

Variable
The name and value of the variable are displayed whenever the value changes. Execution is substantially slower during this form of tracing.

Tracing is turned off whenever the block in which it was turned on is exited. For instance, if the program is stopped inside some procedure and tracing is invoked, the tracing will end when the procedure is exited. To trace the whole program, tracing must be invoked before a run command is issued.

stop at *source-line-number* [**if** *condition*]
stop in *procedure/function* [**if** *condition*]
stop *variable* [**if** *condition*]
stop if *condition*

Stop execution when the given line is reached, procedure or function is called, variable is changed, or condition becomes true. Execution slows considerably when conditional stops are used, and is very slow when checking for a change in a variable.

when in *procedure/function* **{** *command*; ...**}**
when at *source-line-number* **{** *command*; ...**}**
when *condition* **{** *command*; ...**}**

Execute the given *dbx commands* when the procedure or function is called, line number is reached, or condition is true. The braces and the semicolon separating the commands are required.

status [> *filename*]

Display the currently active **trace** and **stop** commands.

delete *command-number* [, ... *command-number*]
delete all

Remove the traces and/or stops corresponding to the given numbers, or all of them. The **status** command displays the numbers associated with traces and stops.

clear *source-line-number*

Clears ALL breakpoints at the given source line number.

catch *number* [, ... *number*]
ignore *number* [, ... *number*]

Start ("catch") or stop ("ignore") trapping the signals with the given *numbers* before they are sent to the program being debugged. This is useful when a program being debugged handles signals such as interrupts. Initially all signals are trapped except SIGHUP, SIGCONT, SIGCHILD, SIGALRM, SIGKILL, SIGSTP, and SIGWINCH.

step [*n*]

Execute through the next *n* source lines and stop on the (*n+1*)st such line. If *n* is not specified, it is taken to be one. Step into procedures and functions.

next [*n*]

Execute through the next *n* source lines and stop on the (*n+1*)st such line. If *n* is not specified, it is taken to be one. Count procedure and function calls as single statements.

When using *conditions* with the **trace** and **stop** commands, remember that variable names are resolved with respect to the current scope not the scope of the **trace** or **stop** command. For example, if you are currently stopped in function **foo** and you issue the command **stop in bar if x == 5**, the variable **x** refers to **x** in function **foo** not in function **bar**. The **func** command can be used to changed the scope before issuing a **trace** or **stop** command.

2. Naming, Printing and Displaying Data

print *expression* [**,** *expression* ...]

Print the values of the expressions. Expressions may involve function calls. If execution encounters any breakpoints, execution halts and the *dbx* command level is re-entered. A stack trace with the **where** command shows that the call originated from the *dbx* command level.

Variables having the same identifier as one in the current block may be referenced as '*block-name***.***variable*', where *block-name* is a unique identifier for a block. For example, to access variable 'c' inside routine 'a' which is declared inside module 'a.c', one would have to use **print a.a.c** to make the name 'a' unambiguous. Use **whereis** to determine the fully qualified name of an identifier.

display [*expression* [**,** *expression* ...]]

Display the values of the expressions each time execution of the program being debugged stops. In *dbxtool*, the expressions and their values appear in the *display* window. The name qualification rules for *print* apply to *display* as well. With no arguments, the *display* command prints a list of the expressions currently being displayed.

undisplay *expression* [**,** *expression* ...]

Stop displaying the expressions and their values each time execution of the program being debugged stops. The name qualification rules for *print* apply to *undisplay* as well.

whatis *identifier*
whatis *type*

Print the declaration of the given identifier or type. The identifier may be qualified with block names as above. *Types* are useful to print all the members of a structure, union, or enumerated type.

which *identifier*

Print the fully qualified form of the given identifier, that is, the outer blocks that the identifier is associated with.

whereis *identifier*
Print the fully qualified form of all the symbols whose names match the given identifier. The order in which the symbols are displayed is not meaningful.

assign *variable = expression*
set *variable = expression*
Assign the value of the expression to the variable. Currently no type conversion takes place if operands are of different types.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function. If execution encounters any breakpoints, execution halts and the *dbx* command level is re-entered. A stack trace with the **where** command shows that the call originated from the *dbx* command level.

If the source file that the routine is defined in was compiled with the -g flag, the number and types of parameters must match. However, if C routines are called which are not compiled with the -g flag, *dbx* will not do parameter checking; the parameters are simply pushed onto the stack as given in the parameter list. Currently, string and structure parameters are not passed properly for C and parameters greater than four bytes in length (for example, complex and double precision variables) and alternate return points are not passed properly for FORTRAN.

where [*n*]
Display a list of the top *n* active procedures and functions on the stack. If *n* is not specified, it displays all active procedures and functions.

dump [*func*]
Display the names and values of all the local variables and parameters in *func*. If *func* is not specified the current function is used.

up [*n*] Move up (towards "main") the call stack *n* levels. If *n* is not specified, the default is one. This command allows you to examine the local variables in functions other than the current one. In *dbxtool*, the line containing the call is hilighted for one second.

down [*n*]
Move down (towards the current stopping point) the call stack *n* levels. If *n* is not specified, the default is one. In *dbxtool*, the line containing the call is hilighted for one second.

3. Accessing Source Files & Directories

edit [*filename*]
edit *procedure/function-name*
Invoke an editor on *filename* or the current source file if none is specified. If a *procedure* or *function* name is specified, the editor is invoked on the file that contains it. Which editor is invoked by default depends on the installation. The default can be overridden by setting the environment variable **EDITOR** to the name of the desired editor.

file [*filename*]
Change the current source file to *filename*. Print the name of the current source file if no *filename* is specified.

func [*procedure/function/program name*]
Change the current function. Print the current function if none is specified. Changing the current function implicitly changes the current source file variable **file** to the one that contains the function; it also changes the current scope used for name resolution. If

the entire scope of the program is desired, the argument should be the object file name.

list [*source-line-number* [**,** *source-line-number*]]
list *procedure/function*

List the lines in the current source file from the first line number through the second. If no lines are specified, the next **10** lines are listed. If the name of a procedure or function is given lines *n*−5 to *n*+5 are listed where *n* is the first statement in the procedure or function. If the **list** command's argument is a procedure or function, the scope for further listing is changed to that routine — use the **file** command to change it back.

use [*directory-list*]

Set the list of directories to be searched when looking for source files. Print the current list of directories if no *directory-list* is given.

cd [*dirname*]

Change *dbx*'s notion of the current directory to *dirname.* With no argument, use the value of the HOME environment variable.

pwd Print *dbx*'s notion of the current directory.

/*string*[**/**]

Search down the current file for the regular expression *string.* The search begins with the line immediately after the current line and, if necessary, continues until the end of the file. The matching line becomes the current line. In *dbxtool,* the matching line is hilighted for one second.

?*string*[**?**]

Search up the current file for the regular expression *string.* The search begins with the line immediately before the current line and, if necessary, continues until the top of the file. The matching line becomes the current line. In *dbxtool,* the matching line is hilighted for one second.

When *dbx* is searching for a source file, the value of 'file' and the 'use' directory search path is used. The value of 'file' is appended to each directory in the 'use' search path until a file is found. This file becomes the current file.

Dbx knows the same file names that are given to the compilers. For instance, if a file is compiled with the command % cc -c -g ../mip/scan.c then *dbx* knows of the file **../mip/scan.c**, but it does not know of the file **scan.c**.

4. Machine-Level Commands

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*variable*] [**if** *cond*]
stopi [**at**] [*address*] [**if** *cond*]

Turn on tracing or set a breakpoint at a machine instruction address.

stepi
nexti

Single step as in **step** or **next**, but do a single machine instruction rather than a source line.

address **,***address***/** [*mode*]
[*address*] **/** [*count*] [*mode*]

Display the contents of memory starting at the first *address* and continuing up to the second *address* or until *count* items have been displayed. If no address is specified, the address following the one displayed most recently is used. The *mode* specifies how memory is to be displayed; if it is omitted the last previously specified mode is used. The initial mode is 'X'. The following modes are supported:

i	display the machine instruction
d	display a word in decimal
D	display a longword in decimal
o	display a word in octal
O	display a longword in octal
x	display a word in hexadecimal
X	display a longword in hexadecimal
b	display a byte in octal
c	display a byte as a character
s	display a string of characters terminated by a null byte
f	display a single precision real number
g	display a double precision real number

Symbolic addresses used in this context are specified by preceding a name with an '&'. Registers are denoted by '$d0-$d7' (data registers), '$a0-$a7' (address registers), '$fp' (frame pointer), '$sp' (stack pointer), and '$pc' (program counter). Note that $fp is equivalent to register a6 and $sp is equivalent to register a7. For example, to print the contents of all registers in hex, one would type "&$d0/20X" or "&$d0,&$a7/X". To print the contents of register d0, one could also type "print $d0" (one cannot currently specify a range with **print**). Addresses may be expressions made up of other addresses and the operators '+', '–', '*', and indirection (unary '*').

5. Miscellaneous Commands

sh *command-line*

Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

alias *new-command-name character-sequence*

Respond to *new-command-name* as though it were *character-sequence*. Special characters occurring in *character-sequence* must be enclosed in quotation marks. Alias substitution ala the C shell (*csh*(1)) also occurs.

help [*command*]

help Print a short message explaining *command*. If no argument is given, display a synopsis of *dbx* commands.

source Read *dbx* commands from the given *filename*. This is especially useful when the *filename* has been created by redirecting a **status** command from an earlier debugging session.

quit Exit *dbx*.

dbxenv stringlen *num*

dbxenv

Set *dbx* attributes. The only one currently supported is **stringlen** which controls the maximum number of characters printed for a "char *" variable in a C program. The **dbxenv** command with no argument prints the attributes and their current values.

debug [*objfile* [*corefile*]]

Terminate debugging of the current program (if any) and begin debugging the one found in *objfile*, without incurring the overhead of reinitializing *dbx*. If no arguments are specified, the name of the program currently being debugged and its arguments is printed.

kill Terminate debugging of the current program, but leave *dbx* ready to debug another. This is particularly useful to eliminate remains of a window program one was debugging without exiting the debugger or to allow the object file to be removed and remade without incurring a "text file busy" error message.

Debugging Processes that Fork

Debugging a process that *fork*(2)s introduces unique problems. *Dbx* uses the *ptrace*(2) interface to fetch from and store into the debuggee. This interface requires that the debugger be the parent process of the debuggee. If the debuggee forks, the child of the fork is the grandchild of *dbx* and it cannot be debugged. No breakpoints should be set in the code that is executed by the child of the fork. If the child of the fork trips over a breakpoint, it will die.

When the debuggee forks, it can still be debugged with a few caveats. After the fork, there are two processes that are sharing the same text (code) space. The kernel does not allow *ptrace* to write into a text space that is being used by more than one process. This means the debuggee should not trip over any breakpoints while the child of the fork is still sharing its text space. In most cases, the child of the fork will *exec*(2) a new program almost immediately. After the exec, it is safe for the debuggee to trip over breakpoints. Therefore, it is recommended that a *sleep*(2) of two or three seconds be placed in the debuggee immediately after the fork. This gives the child of the fork some time to exec a new program and get out of the way.

FILES

a.out	default object file
core	default core file
~/.dbxinit	initial commands

SEE ALSO

cc(1)	C compiler
f77(1)	FORTRAN compiler
pc(1)	Pascal compiler

COMMENTS

Dbx suffers from a 'multiple include' malady. If a program consists of a number of object files and each is built from source files that include header files, the symbolic information for the header files is replicated in each object file. Since roughly debugger start-up is done for each link, having the linker (ld) reorganize the symbol information won't save much time, though it does reduce the disk space used. The problem is an artifact of the unrestricted semantics of #include's in C; for example an include file can contain static declarations that are separate entities for each file in which they are included.

BUGS

Dbx does not correctly handle C variables that are local to compound statements. When printing these variables it often gives incorrect results.

Dbx does not handle FORTRAN entry points well — it treats them as if they were independent routines.

Dbx does not handle *assigning* to FORTRAN complex types correctly (see the *assign/set* command).

The *use* command and -I option do not override the current directory if there is source by the same name in the current directory.

Some operations behave differently in *dbx* than in C.

- *Dbx* has two division operators — **/** always yields a floating point result and **div** always yields an integral result.
- An array or function name does not imply the address of the array or function in *dbx*. An array name implies the entire array, and a function name implies a call to the function with no arguments. The address of an array can be obtained by taking the address of the array's first element, and the address of function can be obtained by taking the address of the function's name.

Casts do not work with FORTRAN or Pascal.

String and structure parameters are not passed properly for C and parameters greater than four bytes in length (for example, complex and double precision variables) and alternate return points are not passed properly for FORTRAN.

NAME

dbxtool — window- and mouse-based debugging tool

SYNOPSIS

dbxtool [**—i**] [**—I** *dir*] [*objfile* [*corefile*]]

DESCRIPTION

Dbxtool is a source-level debugger with a window and mouse-based user interface. It accepts the same commands as *dbx* (1), but provides a more convenient user interface. Using the mouse, one can set breakpoints, examine the values of variables, control execution, peruse source files and so on. *Dbxtool* has separate subwindows for viewing source code, entering commands and several other uses. It functions in the *suntools*(1) environment, so that the standard tool manager actions, such as moving, stretching, exposing etc. can be applied to it.

Objfile is an object file produced by *cc* (1), *f77* (1) or *pc* (1) (or a combination of them) with the appropriate flag (**—g**) specified to produce symbol information in the object file. IMPORTANT: every stage of the compilation process, including the loading phase, must include the **—g** option. If no *objfile* is specified, one uses the *debug* command to specify the program to be debugged. The object file contains a symbol table which includes the names of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named *core* exists in the current directory or a *corefile* is specified, *dbxtool* can be used to examine the state of the program when it faulted.

Debugger commands in the file *.dbxinit* are executed immediately after the symbolic information is read, if that file exists in the current directory, or in the user's home directory if *.dbxinit* doesn't exist in the current directory.

OPTIONS

—i Force *dbxtool* to act as though standard input were a terminal.

—I *dir* Add *dir* to the list of directories that are searched when looking for a source file. Normally *dbxtool* looks for source files in the current directory and then in the directory where *objfile* is located. The directory search path can also be set with the **use** command. Multiple **—I** options may be given.

SUBWINDOWS

Dbxtool consists of five subwindows. From top to bottom they are:

status — the location of the point at which execution is currently stopped and a description of the lines displayed in the *source* subwindow are displayed in the *status* subwindow.

source — displays source text of the program being debugged.

buttons — software buttons corresponding to the most frequently used commands are provided in the *buttons* subwindow. Picking a button with the mouse invokes the command associated with that button.

command — provides a typing interface to *dbxtool* to supplement the *buttons* subwindow and mouse. Also, most of the output generated by commands appears in this subwindow.

display — provides a way of tracking the values of selected variables by updating a display of their values each time execution stops.

SCROLLING

The *source*, *command*, and *display* windows have scroll bars to facilitate browsing their contents. The scroll bar is at the left edge of each window. The bar is a medium gray background with a darker gray area superimposed over it indicating the portion of the source file, command transcript or display currently visible in the window. Note that the size of the darker gray area corresponds to the number of characters visible in the *source* window, not the number of lines.

Within the scroll bar, the mouse buttons have the following functions:

left	scroll forward, move towards the end of the file
middle	scroll to absolute position in the text
right	scroll backwards, move towards the beginning of the file

Positioning the cursor within the scroll bar next to a given line and clicking the left button causes the line to move to the top of the window. Clicking the right button causes the top line in the window to move to the position of the cursor.

The middle button treats the scroll bar as a thumb bar. The top of the thumb bar represents the beginning of the text and the bottom represents the end of the text. Clicking the middle button in the scroll bar picks a point within the text relative to its entire size. This point is displayed at the top of the window.

THE SOURCE WINDOW

The *source* window displays the text of the program being debugged. Initially, it displays text from either the main routine, if there is no core file, or the point at which execution stopped, if there is a core file. Whenever execution stops during a debugging session, it displays the point at which it stopped. The **file** command can be used to switch the *source* window to another file; the focus of attention moves to the beginning of the new file. Similarly, the **func** command can be used to switch the *source* window to another function; the new focus of attention is the first executable line in the function.

Breakpoints are indicated in the *source* window by a solid stop sign at the beginning of the line. The point at which execution is currently stopped is marked by a rightward pointing outlined arrow.

COMMAND CONSTRUCTION

One can either type commands to *dbxtool* or construct them with the selection and button mechanism (if a button is provided for the command), but typing and buttons cannot be combined.

A selection is made by pointing the mouse at one end of the desired text and clicking the left button and then pointing the mouse at the other end of the text and clicking the middle button. The selected text is hilighted in reverse-video. In the *command* window, pressing the right button brings up a "stuff" menu, which, when chosen, causes the selected text to be stuffed into the *command* window's input stream as if it had been typed from the keyboard. Holding down the SHIFT key and clicking the right button is an accelerator that automatically stuffs the selection without bringing up a menu.

The software buttons operate in a postfix manner. That is, one first selects the arguments and then clicks the software button with the left mouse button. Each command interprets the selection as appropriate for that command.

There are five ways in which *dbxtool* may interpret a selection:

literal — A selection may be interpreted as representing exactly the selected material.

expand — A selection may be interpreted as representing exactly the selected material, with the exception that it will be expanded if either the first or last character of the selection is an alphanumeric character or underscore. It will be expanded to the longest enclosing sequence of alphanumeric characters or underscores. Selections made outside of *dbxtool* cannot be expanded and are interpreted as exactly the selected text.

lineno — A selection in the *source* window may be interpreted as representing the (line number of the) first source line containing all or some of the selection.

command — A selection in the *command* window may be interpreted as representing the command containing the selection.

ignore Buttons may ignore a selection.

COMMAND BUTTONS

The standard set of command buttons in the *buttons* window is as follows.

print Prints the value of a variable or expression. It uses the whole selection, except that if the selection string begins or ends with an alphanumeric character or underscore, the string is first expanded from both ends to the longest enclosing string containing only alphanumerics and underscores. Thus, an identifier can be printed by selecting as little as one character within it.

next Executes one source line and then stops execution, except that if the current source line is a procedure or function call it executes through the called routine before stopping. The **next** button ignores the selection.

step Executes one source line and then stops execution again. If the current source line is a procedure or function call it stops at the first executable line within the procedure or function. The **step** button ignores the selection.

stop at Sets a breakpoint at a given source line. It interprets a selection in the *source* window as representing the line number associated with the first line of the selection.

cont Resumes execution from the point at which it is currently stopped. The **cont** button ignores the selection.

stop in Sets a breakpoint at the first line of a given function or procedure. It interprets the selection in the same manner as the **print** button, i.e. selecting an occurrence of a procedure or function name causes a breakpoint to be set in the corresponding routine.

redo Causes a selected command to be repeated. It interprets a selection in the *command* window as representing the command containing the selection.

CHOOSING YOUR OWN BUTTONS

The **button** command may be used to define command buttons in the *buttons* window. It may be used in *.dbxinit* to define buttons in addition to the default set of buttons, or during a debugging session to add new buttons. The first argument to **button** is the selection interpretation to be used for the button and the remainder is the command to be associated with it. The default set of buttons can be replicated by the sequence

```
button expand print
button ignore next
button ignore step
button lineno stop at
button ignore cont
button expand stop in
button command redo
```

The **unbutton** command my be used in *.dbxinit* to remove a default button from the *buttons* window, or during a debugging session to remove any existing buttons. The arguments to the **unbutton** command are the same as for the **button** command — a selection interpretation followed by a command.

THE DISPLAY WINDOW

The *display* window is used to provide continual feedback of the values of a selected set of variables. The **display** command specifies variables to appear in the *display* window and **undisplay** removes them. Each time execution of the program being debugged stops, the values of the displayed variables are updated.

CONTROLLING THE DBXTOOL ENVIRONMENT

The **toolenv** command provides control over several facets of the *dbxtool* window environment, including the font, the vertical size of the *source*, *command* and *display* windows, the horizontal size of the tool, and the minimum number of lines between the top or bottom of the *source* window and the arrow. These are chiefly useful in the *.dbxinit* file to control initialization of the tool, but may be issued at any time.

OTHER ASPECTS OF DBXTOOL

The commands, expression syntax, scope rules, etc. of *dbxtool* are identical to those for *dbx* (1). Three of the commands, **toolenv**, **button**, and **unbutton** affect only *dbxtool*, and so they are described below. See the *dbx* (1) manual entry for a description of the others.

toolenv *attribute value*

Set attributes of the *dbxtool* window. This command has no effect in *dbx*. The **toolenv** command with no arguments prints the current values of the attributes. The possible *attribute-value* pairs and their interpretations are as follows:

font *fontfile*

change the font to that found in *fontfile*; default is taken from the DEFAULT_FONT shell variable

width *nchars*

change the width of the tool window to *nchars* characters; default is 80 characters

srclines *nlines*

make the source subwindow *nlines* high; default is 20 lines

cmdlines *nlines*

make the command subwindow *nlines* high; default is 12 lines

displines *nlines*

make the display subwindow *nlines* high; default is 3 lines

topmargin *nlines*

keep the line with the arrow at least *nlines* from the top of the source subwindow; default is 3 lines

botmargin *nlines*

keep the line with the arrow on it at least *nlines* from the bottom of the source subwindow; default is 3 lines

button *selection command-name*

Associate a button with a command in *dbxtool*. This command has no effect on *dbx*. *Selection* is described in the *Command Construction* section above.

unbutton *selection command-name*

Remove a button from the *buttons* window. The button with a matching *selection* and *command-name* is removed.

FILES

a.out	default object file
core	default core file
~/.dbxinit	initial commands

SEE ALSO

cc(1)	C compiler
f77(1)	FORTRAN compiler

pc(1) Pascal compiler

COMMENTS

The comments listed for *dbx* (1) apply to *dbxtool* as well.

BUGS

The bugs listed for *dbx* (1) apply to *dbxtool* as well.

The selection mechanism of *ttytool* and *dbxtool* should be the same. *Dbxtool* does not have multi-clicking selections.

The interaction between scrolling in the *source* subwindow and *dbx*'s regular expression search commands is wrong. Scrolling should affect where the next search begins, but it does not.

NAME

dc − desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

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

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

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

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

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

d The top value on the stack is duplicated.

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

f All values on the stack and in registers are printed.

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

x treats the top element of the stack as a character string and executes it as a string of dc commands.

X replaces the number on the top of the stack with its scale factor.

[...] puts the bracketed ascii string onto the top of the stack.

<*x* >*x* =*x*
: The top two elements of the stack are popped and compared. Register *x* is executed if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! interprets the rest of the line as a UNIX command.

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input. **I** pushes the input base on the top of the stack.

o The top value on the stack is popped and used as the number radix for further output.

O pushes the output base on the top of the stack.

k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

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

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

; : are used by *bc* for array operations.

EXAMPLE

Print the first ten values of n!

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

SEE ALSO

bc(1), which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

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

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

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

OPTION	*VALUES*
if=*name*	input file is taken from *name;* standard input is default.
of=*name*	output file is taken from *name*; standard output is default. Note that *dd* creates and explicit output file; therefore the **seek** option is usually useless with explicit output except in special cases such as using magnetic tape or raw disk files.
ibs=*n*	input block size *n* bytes (default 512).
obs=*n*	output block size *n* bytes (default 512).
bs=*n*	set both input and output block size, superseding **ibs** and **obs**; also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
files=*n*	copy *n* input files before terminating (makes sense only when input is a magtape or similar device).
seek=*n*	seek *n* records from beginning of output file before copying. This option generally only works with magnetic tapes and raw disk files and is otherwise usually useless if the explicit output file was named with the **of** option.
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
block	convert variable length records to fixed length
unblock	convert fixed length records to variable length
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k** (kilobytes) to specify multiplication by 1024, **b** (blocks of 512 bytes) to specify multiplication by 512, or **w** (words) to specify multiplication by 4; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if **ascii**, **unblock**, **ebcdic**, **ibm**, or **block** conversion is specified. In the first two cases, **cbs** characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and new-line added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size **cbs**.

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

EXAMPLE

To read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x:*

```
tutorial% 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), tr(1)

DIAGNOSTICS

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

BUGS

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

The **block** and **unblock** options cannot be combined with the **ascii**, **ebcdic** or **ibm**. Inalid cobinations silently ignore all but the last mutually-exclusive keyword.

NAME

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

SYNOPSIS

/usr/sccs/delta [**−r** *SID*] [**−s**] [**−n**] [**−g** *list*] [**−m**[*mrlist*]] [**−y**[*comment*]] [**−p**] file …

DESCRIPTION

Delta permanently introduces into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

Delta may issue prompts on the standard output depending upon certain options specified and flags (see *admin*(1)) that may be present in the SCCS file (see **−m** and **−y** options below).

OPTIONS

Options apply independently to each named file.

−r *SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this option is necessary only if two or more outstanding *get*'s for editing (**get −e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **−r** option can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

−s Do not display the created delta's SID, number of lines inserted, deleted and unchanged in the SCCS file.

−n Retain the edited *g-file* which is normally removed at completion of delta processing.

−g *list* Specifies a *list* of deltas to be *ignored* when the file is accessed at the change level (SID) created by this delta. See *get*(1) for the definition of *list*.

−m[*mrlist*]

If the SCCS file has the **v** flag set (see *admin*(1)), a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

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

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

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that the MR numbers were not all valid).

−y[*comment*]

Arbitrary text to describe the reason for making the delta. A null string is considered a valid *comment*.

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

−p Display (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*-file are explained in the *Source Code Control System User's Guide.* The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/bin/diff	Program to compute differences between the "gotten" file and the *g-file*.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (−) is specified on the *delta* command line, the **−m** (if necessary) and **−y** options *must* also be present. Omission of these options is an error.

SEE ALSO

sccs(1), admin(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

deroff – remove nroff, troff, tbl and eqn constructs

SYNOPSIS

deroff [**−w**] file **. . .**

DESCRIPTION

Deroff reads each file in sequence and removes all *nroff* and *troff* command lines, backslash constructions, macro definitions, *eqn* constructs (between '.EQ' and '.EN' lines or between delimiters), and table descriptions and writes the remainder on the standard output. *Deroff* follows chains of included files ('.so' and '.nx' commands); if a file has already been included, a '.so' is ignored and a '.nx' terminates execution. If no input file is given, *deroff* reads from the standard input file.

OPTIONS

−w Generate a word list, one word per line. A 'word' is a string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed. All other characters are ignored.

SEE ALSO

troff(1), eqn(1), tbl(1)

BUGS

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

Deroff does not work well with files that use **.so** to source in the standard macro package files.

NAME

des – encrypt/decrypt with Data Encryption Standard

SYNOPSIS

des –e ¦ –d [–b] [–f] [–k *key* **] [–s] [** *infile* **[** *outfile* **]]**

DESCRIPTION

Des encrypts and decrypts data using the NBS Data Encryption Standard algorithm. One of **-e** (for encrypt) or **-d** (for decrypt) must be specified.

The *des* command is provided to promote secure exchange of data in a standard fashion.

OPTIONS

–b Select ECB (one byte at a time) encryption mode.

–f Suppress warning message when software implementation is used.

–k *key* Use the encryption *key* specified.

–s Selects software implementation for the encryption algorithm.

Two standard encryption modes are supported by the *des* program, Cipher Block Chaining (CBC - the default) and Electronic Code Book (ECB - specified with **-b**). CBC mode treats an entire file as a unit of encryption, i.e., if insertions or deletions are made to the encrypted file then decryption will not succeed. CBC mode also ensures that regularities in clear data do not appear in the encrypted data. ECB mode treats each 8 bytes as units of encryptions, so if parts of the encrypted file are modified then other parts may still be decrypted. Identical values of clear text encrypt to identical values of cipher text.

The key used for the DES algorithm is obtained by prompting the user unless the **-k** *key* option is given. If the key is an argument to the *des* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *des* takes care to destroy the key argument immediately upon entry.

The *des* command attempts to use DES hardware for its job, but will use a software implementation of the DES algorithm if the hardware is unavailable. Normally, a warning message is printed if the DES hardware is unavailable since the software is only about 1/50th as fast. However, the **-f** option will suppress the warning. The **-s** option may be used to force use of software instead of hardware DES.

The *des* command reads from standard input unless *infile* is specified and writes to standard output unless *outfile* is given.

The following sections give information required to implement compatible facilities in other environments.

Since the CBC and ECB modes of DES require units of 8 bytes to be encrypted, files being encrypted by the *des* command have 1 to 8 bytes appended to them to cause them to be a multiple of 8 bytes. The last byte, when decrypted, gives the number of bytes (0 to 7) which are to be saved of the last 8 bytes. The other bytes of those appended to the input are randomized before encryption. If, when decrypting, the last byte is not in the range of 0 to 7 then either the encrypted file has been corrupted or an incorrect key was provided for decryption and an error message is printed.

The DES algorithm requires an 8 byte key whose low order bits are assumed to be odd-parity bits. The ASCII key supplied by the user is zero padded to 8 bytes and the high order bits are set to be odd-parity bits. The DES algorithm then ignores the low bit of each ASCII character, but that bit's information has been preserved in the high bit due to the parity.

The CBC mode of operation always uses an initial value of all zeros for the initialization vector, so the first 8 bytes of a file are encrypted the same whether in CBC or ECB mode.

FILES

/dev/des?

SEE ALSO

des(4)

RESTRICTIONS

This program is not available on software shipped outside the U.S.

NAME

df – report free disk space on file systems

SYNOPSIS

df [**–i**] [filesystem ...] [file ...]

DESCRIPTION

Df displays the amount of disk space occupied by currently mounted file systems, the amount of used and available space, and how much of the file system's total capacity has been used. Used without arguments, *df* produces something like:

```
% df
Filesystem  kbytes  used  avail  capacity  Mounted on
/dev/ip0a    7445    4714 1986   70%       /
/dev/ip0g   42277   35291 2758   93%       /usr
%
```

Note that used+avail is less than the amount of space in the file system (kbytes); this is because the system reserves a fraction of the space in the file system to allow its file system allocation routines to work well. The amount reserved is typically about 10%; this may be adjusted using *tunefs*(8). When all the space on a file system except for this reserve is in use, only the super-user can allocate new files and data blocks to existing files. When a file system is overallocated in this way, *df* may report that the file system is more than 100% utilized.

If arguments to *df* are disk partitions (for example, */dev/ip0a*) or UNIX path names, *df* produces a report on the file system containing the named file. Thus "df ." shows the amount of space on the file system containing the current directory.

OPTIONS

–i Report the number of used and free inodes.

FILES

/etc/mtab list of currently mounted filesystems

SEE ALSO

du(1), mtab(5), icheck(8), quot(8), tunefs(8)

NAME

diff − differential file and directory comparator

SYNOPSIS

diff [**−cefh**] [**−b**] file1 file2
diff [**−D***string*] [**−b**] file1 file2
diff [**−l**] [**−r**] [**−s**] [ ] [**−S***name*] [**−cefh**] [**−b**] dir1 dir2

DESCRIPTION

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

There are several options for output format; the default output format contains lines of these forms:

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

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

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

If both arguments are directories, *diff* sorts the contents of the directories by name, and then runs the regular file *diff* program as described above on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed.

OPTIONS

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

−e produce a script of *a, c* and *d* commands for the editor *ed,* which will recreate *file2* from *file1*. In connection with **−e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo ´1,$p´) | ed − $1
```

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

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

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

−h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

Options for the second form of *diff* are as follows:

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

—b ignore trailing blanks (spaces and tabs); other strings of blanks compare equal.

Options when comparing directories are:

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

—r apply *diff* recursively to common subdirectories encountered.

—s report files which are the same, which are otherwise not mentioned.

—S*name*
starts a directory *diff* in the middle, beginning with file *name.*

FILES

/tmp/d?????
/usr/lib/diffh for **—h**
/usr/bin/pr

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff3(1)

DIAGNOSTICS

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

BUGS

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

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

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**−ex3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

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

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

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

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

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

```
(cat script; echo ´1,$p´) | ed – file1
```

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(1)

BUGS

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

NAME

domainname – set or display name of current domain system

SYNOPSIS

domainname [nameofdomain]

DESCRIPTION

With no arguments, *domainname* displays the name of the current domain. The yellow pages use the domain to refer to a group of hosts.

Only the super-user can set the domainname by giving an argument; this is usually done in the startup script */etc/rc.local.*

SEE ALSO

getdomainname(2), setdomainname(2), ypinit(8)

NAME

du − summarize disk usage

SYNOPSIS

du [**−s**] [**−a**] [name **...**]

DESCRIPTION

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

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

OPTIONS

−s Only display the grand total.

−a Generate an entry for each file.

Entries are generated only for each directory in the absence of options.

EXAMPLE

Here is an example of using *du* in a directory. We used the *pwd* command to identify the directory, then used *du* to show the usage of all the subdirectories in that directory. The grand total for the directory is the last entry in the display:

```
% pwd
/usr/henry/misc
% du
5       ./jokes
33      ./squash
44      ./tech.papers/lpr.document
217     ./tech.papers/new.manager
401     ./tech.papers
144     ./memos
80      ./letters
388     ./window
93      ./messages
15      ./useful.news
1211    .
%
```

SEE ALSO

df(1), quot(8)

BUGS

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

NAME

echo – echo arguments

SYNOPSIS

echo [**—n**] [argument **...**]

DESCRIPTION

Echo writes its arguments on the standard output. Arguments must be separated by spaces or tabs, and terminated by a newline.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. If you are using the Bourne Shell (*sh*(1)), you can send diagnostics to the standard error file by typing: 'echo **...** 1>&2'.

OPTIONS

—n Don't add the newline to the output.

NAME

ed — text editor

SYNOPSIS

ed [**–**] [**–x**] [filename]

DESCRIPTION

Ed is the basic text editor in the UNIX system. While *ed* is for all practical purposes superseded by *vi, ed* is still used by other system utilities such as SCCS. *Ed* is a line editor — in general, you must specify the line or lines on which to perform the operation you choose (see *Line Addressing*, below, for a discussion of how to form line-addresses for *ed*). You can tell *ed* to perform various operations on the lines. For instance, you can print (display) lines; you can change lines; you can insert new lines into the buffer; you can delete existing lines; you can move or copy lines to a different place in the buffer; you can substitute character strings within lines. See *List of Operations*, below, for a guide. Also, see *Regular Expressions* for string-matching metacharacters.

Ed does not directly change the contents of a file — when editing a file, *ed* reads the contents of the file into a *buffer* or *scratchpad*. All changes made during an editing session are made on the contents of this buffer. The copy must be 'saved' or 'written' — using the *w* (**write**) operation — to save changes.

The command-line shown in the synopsis above invokes *ed*. If *filename* is given, *ed* reads a copy of *filename* into its buffer so that it can be edited (simulates an *e* operation on *filename*).

Ed commands have a simple and regular structure: commands consists of an optional line-address, or two optional line-addresses separated by a comma, then a single letter operation, optionally followed by some other parameter:

[*line-address* [*,line-address*]] **operation** [*parameter*]

For example, '**1,10p**' means 'print (display) lines 1 through 10' (two line-addresses), '**5a**' means 'append after line 5' (one line-address), and **d** means 'delete the current line' (no line-address with the current line used as default). *Parameter* varies for each operation — for the **m**ove and **t**ransfer operations, for example, it is the line that the addressed lines are to be **m**oved or **t**ransferred after. These operations actually have three line-addresses. For reading and writing a file, *parameter* specifies the name of the file that is to be read.

Ed is extremely terse in its interaction with the user — *ed*'s normal response to just about any problem is simply a question mark **?**. You get this response when, for instance, *ed* can't find a specified line in the buffer, or if a search for a regular expression fails in a substitute (**s**) operation.

OPTIONS

– Suppress the display of character counts normally given by the **e**, **r**, and **w** operations — can be used when the standard input is an editor script.

–x Simulate an **x** operation on the named file before reading it into the buffer, to handle encryption.

LINE ADDRESSING

The format of *ed* operations above shows that an operation can be preceded by one or two line-addresses, both of which are optional. If only one line-address is specified, operations are performed on that specific line. If two line-addresses are supplied, *ed* operates on the inclusive range of lines between them.

Line-addresses are usually separated from each other by a comma — for instance:

```
1,10p
```

prints (displays) lines 1 thru 10.

Line addresses may also be separated by a semicolon. Whereas the starting position for line-addresses separated by a comma is the same place in the buffer, when a line-address is followed by a semicolon, the current line is set to the line-address preceding the semicolon before any subsequent line-addresses are interpreted. For example:

```
        /Domaine Chandon/;//p
```

sets the current line to the first occurrence of the string 'Domaine Chandon' before starting the search for the second occurrence. This feature can be used to determine the starting line for forward and backward searches ('/', '?').

Lines can be accessed (addressed, in *ed* terminology) in several ways, but the most easily understood way of addressing lines is by line number. Line numbers in *ed* are relative to the start of the buffer. In practice, addressing lines by number proves to be the most awkward to use, so *ed* provides other mechanisms for line-addressing. Note that the line numbers associated with lines in the buffer are not physically present with the text of the lines — they are just an addressing mechanism.

While *ed* is working on the buffer, it keeps track of the line on which you last performed some operation. This line is called the 'current line'. As described below, you can indicate the current line by typing a period character (**.**).

If you don't specify a line for an operation to operate on, most *ed* operations work on the line addressed by the current line.

When *ed* starts working on a file, the current line is positioned at the last line in the buffer. Thereafter, the current line usually changes when any operation is performed. In general, the current line sits at the last line affected by whatever *ed* operation you used. For instance, if you print lines 1 through 10 of the buffer, after the lines are displayed, the current line will be positioned at line 10.

Line-addresses are constructed from elements as shown in the list below. Some special characters are used as a shorthand for certain line-addresses:

. ('dot') addresses the current line.

$ addresses the last line of the buffer.

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

'x addresses the line marked with the name *x*, which must be a lower-case letter. Mark lines with the **k** operation described below.

/regular expression/
A *regular expression* enclosed in slashes '/' searches forward from the current line and stops at the first line containing a string that matches the *regular expression*. If necessary, the search wraps around to the beginning of the buffer.

?regular expression?
A *regular expression* enclosed in question marks '?' searches backward from the current line and stops at the first line containing a string that matches the *regular expression*. If necessary the search wraps around to the end of the buffer.

address±nnn
An *address* followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that line-address plus or minus the indicated number of lines. Plus is assumed if no signs are given.

±address
An *address* beginning with '+' or '−' is taken relative to the current line; in other words, '−5' is understood to mean '**.**−5'.

address±
An *address* ending with '+' or '−', adds or subtracts 1. As a consequence of this rule and

the previous rule, the line-address '–' refers to the line before the current line. Moreover, trailing '+' and '–' characters have cumulative effect, so '– –' refers to the current line less 2.

To maintain compatibility with earlier versions of *ed*, the character '^' in line-addresses is equivalent to '–'.

Ed operations do not necessarily use line-addresses; they may use one or two. Operations which don't use line-addresses regard the presence of a line-address as an error. Operations which accept one or two line-addresses assume default line-addresses if these are not specified. If more line-addresses are given than such an operation requires, the last one or two (depending on what is accepted) are used. The second line-address of any two-address sequence must be greater than the first line-address — that is, the second line must follow the first line in the buffer.

LIST OF OPERATIONS

Ed operates in one of two major modes: ***command mode*** and ***text input mode.*** *Ed* always starts up in command mode.

While you are typing commands at *ed*, you are in command mode. Some commands — **a** for **a**ppend, **c** for **c**hange, and **i** for **i**nsert — provide for adding new text to the buffer. While *ed* is accepting new text, you are in text input mode. You exit from text input mode by typing a period '**.**' alone at the beginning of a line. *Ed* then reverts to command mode. For example, here is a very short illustration of command mode versus text mode:

```
example% ed wineliist       (tell ed to edit a file called winelist)
42                          (ed states there are 42 characters in the file)
1,$p                        (in command mode — tell ed to print all lines)
1978 Chateau Chunder
1979 Redeye Canyon
a                           (in command mode — tell ed to append text)
1980 Doomsday Special       (text input mode — add a new line)
.                           (period ends text input mode)
p                           (back in command mode — print last line entered)
1980 Doomsday Special
w                           (command mode — write the file)
65                          (ed displays the number of characters written)
q                           (command mode — quit the edit session)
example%                    (back in the Shell)
```

If you interrupt *ed*, it displays '?interrupted' and returns to command mode.

a Append Text.

Reads the text entered in input mode and appends it to the buffer after the addressed line. **a** accepts one line-address — default line-address is the current line. The new current line is the last line input, or at the addressed line if no text is entered. Address '0' is a valid place to append text, in which case text is placed at the beginning of the buffer.

c Change Lines.

Deletes the addressed lines, then accepts input text which replaces these lines. **c** accepts two line-addresses — default line-address is the current line. The current line is left on the last line input, or at the line preceding the deleted lines if no text is entered.

d Delete Lines.

Delete the addressed lines from the buffer. **d** accepts two line-addresses — default line-address is the current line. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e *filename* Edit a file.

Deletes the entire contents of the buffer, and then reads in the named file. **e** sets the current line to the last line of the buffer, and reports the number of characters read into the buffer. **e** remembers *filename* for possible use as a default file name in a subsequent **r** or **w** operations. If no *filename* is given, the remembered filename is used. **e** displays a **?** if the buffer has not been written out since the last change made — a second **e** operation says you really mean it.

E *filename*
Same as **e**, but will silently allow you to quit an editing session without warning you if you have not written your file. **e**, on the other hand, reminds you to save your changes if you have altered the buffer at all.

f *filename* Display Remembered Filename.
Display the currently 'remembered filename'. If *filename* is given, the currently 'remembered filename' is changed to *filename*.

g/*regular expression*/*operation list*
This is the **global** operation: perform *operation list* on all lines in the range of line-addresses containing *regular expression*. **g** accepts two line-addresses — default is all lines in the buffer. Also see the **v** operation, which inverts the sense of *regular expression*.

If your *operation list* actually takes up more than a single line, you must end every line except the last (the true 'end' of the global operation) with an escape character, '\'. For example, if you want to substitute 'jimjams' for 'frammis', then append several lines of text to every line containing the string 'widget' and print those lines, you would type this sequence:

```
g/widget/s/frammis/jimjams/\
a \
new line of text\
another new line of text\
.\
p
```

Note that the **a**, **i**, and **c** operations, which put *ed* in input mode, are permitted in the *operation list*; the final **.** terminating input may be omitted if it is the last line of the *operation list*. The **g** and **v** operations are not permitted in the *operation list*.

i Insert Text.
Insert lines of text into the buffer before the addressed line. **i** accepts one line-address — default line-address is the current line. The current line is placed at the last line input; if no text is input, the current line is left at the line before the addressed line. **i** differs from **a** only in the placement of the text.

j Join Lines.
Joins the addressed lines into a single line; intermediate newlines simply disappear. **j** accepts two line-addresses — default is the current line and the following line. The current line is placed at the resulting line.

k*x* Mark Line.
Marks the addressed line with name *x* (the name must be a lower-case letter). The line-address form '*x* then addresses this line. **k** accepts one line-address — default line-address is the current line.

l Display Non-printing Characters.
Displays non-graphic characters in the addressed lines such that they are displayed in two-digit octal, and long lines are folded. **l** accepts two line-addresses — default line-address is the current line. **l** may be placed on the same line after any non-I/O operation.

m*address* Move lines.
Reposition the addressed lines after the line-addressed by *address*. **m** accepts two line-addresses to specify the range of lines to be moved — default line-address is the current line. The last of the moved lines becomes the current line.

p Print (display) Lines.
Displays the addressed lines. **p** accepts two line-addresses — default line-address is the current line. The current line is placed at the last line printed. **p** may be placed on the same line after any non-I/O operation.

P Synonym for **p**.

q Quit Edit Session.
Exit from the editing session. Note, however, that the buffer is not automatically written out (do a '**w**' to write if you want to save your changes). *Ed* warns you once if you haven't saved your file — a second **q** says you really mean it.

Q Same as **q**, but you don't get any warning if you haven't previously written out the buffer.

r *filename* Read from file.
Reads the contents of *filename* into the buffer after the addressed line. If *filename* is not given, the 'remembered filename', if any, is used (see **e** and **f**). **r** accepts one line-address — default line-address is **$**. If line-address '0' is used, **r** reads the file in at the beginning of the buffer. If the read is successful, **r** displays the number of characters read in. The current line is left at the last line read in from the file.

s/*regular expression*/*replacement string*/ or,
s/*regular expression*/*replacement string*/**g**
Substitute the *replacement string* for the first occurrence of *regular expression* on each line where the *regular expression* occurs. In the first form of the **s** operation, only the first occurrence of the matched string on each line is replaced. If you use the **g** (global) suffix, **all** occurrences of the *regular expression* are replaced in the line. Keep the **g** *suffix* of the **s** operation distinct from the **g** operation itself — they are completely different. **s** accepts two line-addresses to delimit the range of lines within which the substitutions should be done — default line-address is the current line. The current line is left at the last line substituted.

Special Characters:
Any punctuation character may be used instead of '/' to delimit the *regular expression* and the *replacement string*.
An ampersand '&' appearing in the *replacement string* is replaced by the string matching the *regular expression*. The special meaning of '&' in this context may be suppressed by preceding it by '\'.
The characters **\***n* where *n* is a digit, are replaced by the text matched by the *n*-th *regular subexpression* enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left. Lines may be split by substituting new-line characters into them. The new-line in the *replacement string* must be escaped by preceding it by '\'.

t*address* Transfer Lines.
Transfers a copy of the addressed lines to after line *address*. **t**ransfer is like **m**ove, but it makes copies of the lines, leaving the original text where it was. **t** accepts two line-addresses preceeding the operation letter — default line-address is default. The current line is left on the last line of the copy. '0' is a legal line-address for the destination.

u Undo. Undo previous substitute.
undo undoes the effect of the the last substitute operation, *providing that the current line has not been moved since the substitute operation*.

v/regular expression/operation list
Like a negative of the global operation, **g**: perform *operation list* on all lines *except* those containing *regular expression*. **v** accepts two line-addresses — default is all lines in the file.

w Write Lines.
Write the addressed lines from the buffer into the file specified by the 'remembered filename'. **w** accepts two line-addresses — default is all lines in the file. The current line is unchanged. If the write is successful, *ed* displays the number of characters written.

w *filename* Write Lines.
Write the addressed lines into *filename*. *Filename* is created if it does not already exist. *Filename* becomes the 'remembered filename' (see the **e** and **f** operations). **w** accepts two line-addresses — default is all lines in the file. The current line is unchanged. If the write is successful, *ed* displays the number of characters written.

W *filename*
Same as **w**, but appends the addressed lines to the named file instead of overwriting the file. **W** accepts two line-addresses — default is all lines in the file.

x Encrypt File.
When **x** is used, *ed* demands a key string from the standard input. Later **r**, **e**, and **w** operations will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

= Display Line Number.
Display the line number of the addressed line. **=** accepts one line-address — default line-address is $. The current line is unchanged by this operation.

!<shell command>
The remainder of the line after the '!' is sent to *sh*(1) to be interpreted as a shell command. The current line is unchanged.

*address***<newllne>**
Display the addressed line. If you type a line-address and type RETURN, *ed* displays the addressed line. If you simply type RETURN, the line following the current line is displayed (equivalent to '.+1p'). This is useful for stepping through text.

REGULAR EXPRESSIONS

Ed supports a limited form of *regular expression* notation. A regular expression (also known as a *pattern*) specifies a set of strings of characters — such as 'any string containing digits 5 through 9' or 'only lines containing uppercase letters'. A member of this set of strings is said to be *matched* by the regular expression. Regular expressions or patterns are used to address lines in the buffer (see *Line Addressing*, above), and also for selecting strings to be substituted in the **s** (substitute) operation described previously.

An empty regular expression, indicated by two regular expression delimiters in a row, stands for a copy of the last regular expression encountered.

Any given regular expression matches the the longest among the leftmost matches in a line.

In the following specification for regular expressions, the notation *c* stands for any single ordinary character, where a character is anything except a newline character.

c any ordinary character except a special character matches itself. Special characters are the delimiters that actually surround the regular expression, plus \ (the escape character), **[** (the opening bracket for a character class as described below), **.** (period which matches any single character), and sometimes the * (closure) ^ and $ characters. If you want a literal occurrence of one of these special characters you must escape them with the \\ character.

^ at the very start of the regular expression constrains the match to the beginning of the line. A match of this type is called an 'anchored match' because it is 'anchored' to a specific place in the line. The ^ character loses its special meaning if it appears in any position other than at the very start of the regular expression.

$** at the very end of the regular expression constrains the match to the end of the line. A match of this type is called an 'anchored match' because it is 'anchored' to a specific place in the line. The **$ character loses its special meaning if it appears in any position other than at the very end of the regular expression.

. (period) matches any single character except a newline character.

[*string***]**
A *string* of characters enclosed in brackets matches any one of the characters in the brackets. For example, [abcxyz] matches any single character from the set 'abcxyz'. If the first character inside the bracket is a ^, the string matches any character **not** inside the brackets. For instance, [^456787] matches any character except '45678'. You can use a shorthand notation to refer to an inclusive *range* of characters: **a—b**. Such a bracketed string of characters is known as a *character class.*

When two regular expressions are concatenated, they match the leftmost and then the longest possible string that can be divided with the first part of the string matching the first regular expression, followed by the second string matching the second regular expression.

***** Any regular expression followed by * matches a sequence of 0 or more matches of the regular expression. Such a pattern is called a *closure*. For example: **[a—z][a—z]*** matches any string of one or more lower case letters.

\(*regular expression* **\)**
The *regular expression* within the **\(** and **\)** brackets essentially 'remembers' whatever the *regular expression* matches. This provides a mechanism for extracting parts of strings. There can be up to nine such partial matches in a string. Parenthesized *regular expressions* can be nested.

\*n* where *n* is in the range 1 thru 9, matches a copy of the string that the bracketed regular expression beginning with the *n*th \(matched, as described in the previous paragraph on matching parts of strings. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left.

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

FILES

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

SEE ALSO

Using the ed Line Editor in *Editing and Text Processing on the Sun Workstation.*

crypt(1)	encode and decode
ex(1)	extended line editor (part of *vi*)
sed(1)	stream editor
vi(1)	visual (display) editor

BUGS

The **l** operation mishandles DEL.
The **undo** operation removes marks from affected lines.
Because 0 is an illegal line-address for a *w* operation, it is not possible to create an empty file

with *ed*. Use *cat*(1) to create an empty file.

RESTRICTIONS

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the temporary file. Since each line uses two bytes of memory, the limit on the number of lines should not be exceeded in practice.

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

The encryption facilities of *ed* are not available on software shipped outside the U.S.

NAME

eqn, neqn, checkeq — typeset mathematics

SYNOPSIS

eqn [**−d***xy*] [**−p***n*] [**−s***n*] [**−f***n*] [*filename*] ...
neqn [*filename*] ...
checkeq [*filename*] ...

DESCRIPTION

Eqn (and *neqn*) are language processors to assist in describing equations. *Eqn* is a preprocessor for *troff*(1) and is intended for devices that can print *troff*'s output. *Neqn* is a preprocessor for *nroff*(1) and is intended for use with terminals. Usage is almost always:

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

If no *files* are specified, *eqn* and *neqn* read from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input.

Checkeq reports missing or unbalanced delimiters and .EQ/.EN pairs.

OPTIONS

−d *xy* Set equation **d**elimiters may be set to characters *x* and *y* with the command-line argument. The more common way to do this is with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off' appearing in the text. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

−p *n* Reduce subscripts and superscripts by *n* point sizes from the previous size. In the absence of the **−p** option, subscripts and superscripts are reduced by 3 point sizes from the previous size.

−s *n* Change point **s**ize to *n* globally in the document. The point size can also be changed globally in the body of the document by using the **gsize** directive.

−f *n* Change **f**ont to *n* globally in the document. The font can also be changed globally in the body of the document by using the **gfont** directive.

EQN LANGUAGE

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

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 +bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n \to \infty} \sum_0^n x_i$ is made with *lim from {n−> inf } sum from 0 to n x sub i.*

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~=~1* produces $\left[x^2 + \frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional.

Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x}=\overline{f(t)}$, *y dotdot bar ~=~ n under* is $\bar{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \overleftrightarrow{y}$.

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

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

Shorthands may be defined or existing keywords redefined with **define**:

define thing % replacement %

defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

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

SEE ALSO

troff(1), tbl(1), ms(7), eqnchar(7)
Typesetting Mathematics with 'eqn' and
Formatting Documents with 'nroff' and 'troff'
in *Editing and Text Processing on the Sun Workstation*.

BUGS

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

NAME

error — analyze and disperse compiler error messages

SYNOPSIS

error [**−n**] [**−s**] [**−q**] [**−v**] [**−t** *suffixlist*] [**−I** *ignorefile*] [name]

DESCRIPTION

Error analyzes error messages produced by a number of compilers and language processors. It replaces the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously.

Error looks at error messages, either from the specified file *name* or from the standard input, and:

- Determines which language processor produced each error message,
- Determines the file name and line number of the erroneous line, and,
- Inserts the error message into the source file immediately preceding the erroneous line.

Error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. *Error* touches source files only after all input has been read.

Options are explained later on in this manual entry.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into *error*. For example, when using the *csh* syntax,

```
        tutorial% make -s lint |& error -q -v
```

will analyze all the error messages produced by whatever programs *make* runs when making lint.

Error knows about the error messages produced by: *make*, *cc*, *cpp*, *ccom*, *as*, *ld*, *lint*, *pi*, *pc*, and *f77*. For all languages except Pascal, error messages are restricted to one line. Some error messages refer to more than one line in more than one file, in which case *error* duplicates the error message and inserts it at all of the places referenced.

Error does one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file they are processing. *Error* uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by *error*.

discard Error messages from *lint* that refer to one of the two *lint* libraries, */usr/lib/llib-lc* and */usr/lib/llib-port* are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by *error*.

nullify Error messages from *lint* can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named *.errorrc* in the user's home directory, or from the file named by the **−I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
: Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They are not inserted into any source file.

file specific
: Error messages that refer to a specific file but to no specific line are written to the standard output when that file is touched.

true errors
Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are inserted into source files. Other error messages are consumed entirely by *error* or are written to the standard output. *Error* inserts the error messages into the source file on the line preceeding the line number in the error message. Each error message is turned into a one line comment for the language, and is internally flagged with the string '###' at the beginning of the error, and '%%%' at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

OPTIONS

−n Do *not* touch any files; all error messages are sent to the standard output.

−q *Error* asks whether the file should be touched. A 'y' or 'n' to the question is necessary to continue. Absence of the **−q** option implies that all referenced files (except those refering to discarded error messages) are to be touched.

−v After all files have been touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi* can't be found, try *ex* or *ed* from standard places.

−t Take the following argument as a suffix list. Files whose suffices do not appear in the suffix list are not touched. The suffix list is dot seperated, and '*' wildcards work. Thus the suffix list:

".c.y.f*.h"

allows *error* to touch files ending with '.c', '.y', '.f*' and '.y'.

−s Print out *statistics* regarding the error categorization. Not too useful.

Error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

FILES

~/.errorrc	function names to ignore for *lint* error messages
/dev/tty	user's teletype

BUGS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause *error* to not understand the error message.

Error, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the '|' marking the point of error is also disturbed by *error.*

Error was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

NAME

ex, edit — text editor

SYNOPSIS

ex [**–**] [**–R**] [**–r**] [**–t** *tag*] [**+***command*] [**–v**] [**–x**] [**–w***nnn*] [**–l**] *file* ...
edit [ex options]

DESCRIPTION

Ex is the root of a family of editors which includes *edit*(1), *ex*, and *vi*(1). Display based editing is the focus of *vi*. Most users will in fact use *vi* as the principal interface to the system rather than *ex*.

OPTIONS

– supress all interactive feedback to the user — useful for processing *ex* scripts in shell files.

–R edit the file in read-only state.

–r recover the indicated *file*s after a crash.

–t *tag* edit the file containing the tag *tag*. A tags database must have been built previously using the *ctags*(1) command.

+*command*
start the editing session by executing *command*.

–v start up in display editing state using *vi*(1). You can achieve the same effect by simply typing the *vi* command itself.

–x prompt for a key to be used in encrypting the file being edited.

–w*nnn*
set the default window (number of lines on your terminal) to *nnn*— this is useful if you are dialling into the system over a slow 'phone line.

–l set up for editing LISP programs.

DOCUMENTATION

The document *Edit: A tutorial* provides a comprehensive introduction to *edit* assuming no previous knowledge of computers or the UNIX system.

The *Ex Reference Manual – Version 3.5* is a comprehensive and complete manual for the command mode features of *ex*, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed;* the material in the introductory and advanced documents works also with *ex*.

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi*. These documents can be found in the *Editing and Text Processing* Manual. In addition, the *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction*.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), grep(1), vi(1), termcap(5), environ(5)

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.

The editor checks the first five lines of the text file for commands of the form "ex:*command" or "vi:command.*" This feature can result in unexpected behavior, and is not recommended in any case.

RESTRICTIONS

The encryption facilities of *ex* are not available on software shipped outside the U.S.

NAME

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [–tabstop] [–tab1,tab2,...,tabn] [file ...]
unexpand [**–a**] [file ...]

DESCRIPTION

Expand copies the named *files* (or the standard input) to the standard output, with tabs changed into spaces (blanks). Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

Unexpand copies the named *files* (or the standard input) to the standard output, putting tabs back into the data. By default only leading spaces (blanks) and tabs are converted to strings of tabs, but this can be overridden by the **–a** option (see the *options* section below).

EXPAND OPTIONS

–tabstop
Specified as a single argument sets tabs *tabstop* spaces apart instead of the default 8.

–tab1,tab2,...,tabn
Set tabs at the columns specified by *tab1...*

UNEXPAND OPTIONS

–a Insert tabs when replacing a run of two or more spaces would produce a smaller output file. This option only applies to *unexpand.*

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

Expr evaluates expressions as specified by its arguments. Each token of the expression is a separate argument. After evaluation, the result is written on the standard output.

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

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

expr & expr
: yields the first *expr* if neither *expr* is null or '0', otherwise yields '0'.

expr relop expr
: yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expr* are integers, otherwise the comparison is lexicographic. *relop* is one of < (less than), <= (less than or equal to), = (equal to), != (not equal to), >= (greater than or equal to), and > (greater than).

expr + *expr*
expr – *expr*
: addition or subtraction of the arguments.

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

expr : *expr*
match *string regular-expression*
: The two forms of the matching operator above are synonymous. The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of *ed*(1). The **\(...\)** pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

substr *string integer-1 integer-2*
: extracts the subtring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1 has a value greater than string*, *expr* returns a null string. If you try to extract more characters than there are in *string*, *expr* returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as *expr* tends to run forever in these cases.

index *string character-list*
: reports the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

length *string*
: returns the length (that is, the number of characters) of *string*.

(*expr*)
: parentheses for grouping.

EXAMPLES

To add 1 to the Shell variable *a*:

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

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

```
expr $a : ´.*/\(.*\)´ ´|´ $a
```

Note the quoted Shell metacharacters.

SEE ALSO

sh(1), test(1)

DIAGNOSTICS

Expr returns the following exit codes:

0 if the expression is neither null nor '0',
1 if the expression is null or '0',
2 for invalid expressions.

BUGS

Note that the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

```
tutorial% expr index expurgatorious length
syntax error
tutorial%
```

generates the 'syntax error' message as shown instead of the value 1 as you might expect.

NAME

eyacc – modified yacc allowing much improved error recovery

SYNOPSIS

eyacc [**–v**] [grammar]

DESCRIPTION

Eyacc is an old version of *yacc*(1), which produces tables used by the Pascal system and its error recovery routines. *Eyacc* fully enumerates test actions in its parser when an error token is in the look-ahead set. This prevents the parser from making undesirable reductions when an error occurs before the error is detected. The table format is different in *eyacc* than it was in the old *yacc,* as minor changes had been made for efficiency reasons.

SEE ALSO

yacc(1)
Practical LR Error Recovery by Susan L. Graham, Charles B. Haley and W. N. Joy; SIGPLAN Conference on Compiler Construction, August 1979.

BUGS

Pc and its error recovery routines should be made into a library of routines for the new *yacc.*

NAME

f77 – FORTRAN 77 compiler

SYNOPSIS

f77 [**−c**] [**−g**] [**−o** *output*] [**−O** [[**−C** [[**−D***name* [[**−D***name*[**=***def*]] [**−fsky** [[**−F** [
[**−i2** [[**−I***dir*]] **−m** [[**−N**[**qxscn**[*nnn*] [**−onetrip**] [**−p** [[**−pg** [[**−R** *x* [[**−S** [
[**−u**] [**−U**] [**−v** [[**−w**[**66**]] *filename* ...

DESCRIPTION

f77 is the UNIX FORTRAN 77 compiler, which translates programs written in the FORTRAN 77 programming language into executable load modules or into relocatable binary programs for subsequent loading via the *ld*(1) linker. In addition to the many flag arguments (options), *f77* accepts several types of files:

Filenames ending in *.f* are taken to be FORTRAN 77 source programs; they are compiled, and each object program is left in the file (in the current directory) whose name is that of the source with *.o* substituted for *.f*. Filenames ending in *.F* are also taken to be FORTRAN 77 source programs, but they are preprocessed by the C preprocessor (equivalent to a **cc −E** command) before they are compiled by the *f77* compiler.

Filenames ending in *.r* are taken to be Ratfor source programs; these are first transformed by the *ratfor*(1) preprocessor, then compiled by *f77*.

In the same way, filenames ending in *.c* or *.s* are taken to be C or assembly source programs and are compiled or assembled, producing *.o* files.

OPTIONS

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

−c Suppress loading and produce a *.o* file for each source file.

−g Produce additional symbol table information for *dbx*(1). Also pass the **−lg** flag to *ld*(1).

−o *output*
Name the final output file *output* instead of *a.out*.

The following options are peculiar to *f77*:

−O Optimize the object code. This invokes both the global intermediate code optimizer and the object code optimizer.

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

−D*name*

−D*name*=*definition*
Define the *name* to the C preprocessor, as if by '#define'. If no definition is given, the name is defined as "1". (*.F* suffix files only).

−fsky Generate code which assumes the presence of a SKY floating-point processor board. Programs compiled with this option can only be run in systems that have a SKY board installed. Programs compiled without the **−fsky** option will use the SKY board if it is present, but won't run as fast as they would if the **−fsky** option were used. If any part of a program is compiled using the **−fsky** option, you must also use this option when linking with the *f77* command, since a different set of startup routines is required.

−F Apply the C preprocessor to *.F* files and the Ratfor preprocessor to do not compile them. No loading is done. However, any *.c* or *.s* files are compiled or assembled. *.f*, but do not compile.

−i2 Make the default size of integer and logical constants and variables two bytes rather than four bytes.

−I*dir* '#include' files whose names do not begin with '/' are always sought by the C preprocessor first in the directory of the *file* argument, then in directories named in **−I** options, then in the */usr/include/f77* directory (applies to processing of *.F* suffix files only).

−m Apply the M4 preprocessor to each *.r* file before transforming it with the Ratfor preprocessor.

−N[**qxscn**]*nnn*
Make static tables in the compiler bigger. *f77* complains if tables overflow and suggests you apply one or more of these flags. These flags have the following meanings:

q Maximum number of equivalenced variables. Default is 150.

x Maximum number of external names (common block, subroutine, and function names). Default is 200.

s Maximum number of statement numbers. Default is 401.

c Maximum depth of nesting for control statements (for example, DO loops). Default is 20.

n Maximum number of identifiers. Default is 1009.

One may use multiple **−N** options to increase the size of multiple tables.

−onetrip
Compile DO loops so that they are performed at least once if reached. FORTRAN 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.

−p Produce code to count the number of times each routine is called and estimate the fraction of time spent in each routine. The results of this profiling appear in a file called *mon.out* when the program being profiled terminates normally. An execution profile of the program can then be produced using the *prof*(1) utility.

−pg Produce counting code in the manner of **−p,** but invoke a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. An execution profile can then be generated by use of *gprof*(1).

−R*x* Use the string *x* as a Ratfor option in processing *.r* files.

−S Compile the named programs, and leave the assembly language output on corresponding files suffixed *.s* (no *.o* file is created).

−u Make the default type of a variable 'undefined' rather than using the FORTRAN default rules.

−U Do not convert upper case letters to lower case. The default is to convert upper case letters to lower case, except within character string constants.

−v Print the version number of the compiler, and the name of each pass as it executes.

−w[**66**]
Suppress all warning messages, or if the option is **−w66**, supress only FORTRAN 66 compatibility warnings.

Other arguments are taken to be either loader option arguments, or *f77*-compatible object programs, typically produced by an earlier run, or libraries of *f77*-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program in the file specified by the **−o** option, or in a file named *a.out* if the **−o** option is not specified.

FILES

file.[fFrsc]	input file
file.o	object file
a.out	linked output
./fort[pid].?	temporary
/usr/include/f77	directory searched by the FORTRAN 77 include statement

/usr/lib/f77pass1	parser
/lib/f1	code generator
/lib/c2	optional optimizer
/lib/cpp	C preprocessor
/lib/bin/ratfor	Ratfor preprocessor
/usr/lib/libf77.a	Fortran library
/lib/libc.a	C library, see Section 3
mon.out	file produced for analysis by prof(1)
gmon.out	file produced for analysis by gprof(1)

SEE ALSO

FORTRAN Programmer's Guide for the Sun Workstation
prof(1), gprof(1), cc(1), ld(1), ratfor(1)

DIAGNOSTICS

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

NAME

false, true – provide truth values

SYNOPSIS

true

false

DESCRIPTION

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

EXAMPLE

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

SEE ALSO

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

DIAGNOSTICS

False has exit status nonzero.

NAME

file – determine file type

SYNOPSIS

file [**–f**] file **...**

DESCRIPTION

File performs a series of tests on each *file* in an attempt to determine what its contents are. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language.

OPTIONS

–f If the **–f** flag is used, *file* assumes that the filename given next on the command line is a file containing a list of files, and operates on each file listed.

EXAMPLE

The example illustrates the use of *file* on all the files in a specific user's directory:

```
% pwd
/usr/henry/misc
% ls
bensusan        fortran.mss     messages        romero          toolkit.tr
command.list    jokes           pascal.mss      squash          useful.news
counts          letters         play.mss        tech.papers     v7.stuff
deuter          memos           roadmap.mss     titles          window
% file *
bensusan:       roff, nroff, or eqn input text
command.list:   roff, nroff, or eqn input text
counts:         ascii text
deuter:         roff, nroff, or eqn input text
fortran.mss:    roff, nroff, or eqn input text
jokes:          directory
letters:        directory
memos:          directory
messages:       directory
pascal.mss:     roff, nroff, or eqn input text
play.mss:       roff, nroff, or eqn input text
roadmap.mss:    roff, nroff, or eqn input text
romero:         roff, nroff, or eqn input text
squash:         directory
tech.papers:    directory
titles:         ascii text
toolkit.tr:     roff, nroff, or eqn input text
useful.news:    directory
v7.stuff:       archive
window:         directory
%
```

BUGS

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

Does not recognize Pascal or LISP.

NAME

find – find files

SYNOPSIS

find pathname-list expression

DESCRIPTION

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

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

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

−type c True if the type of the file is *c*, where *c* is **b, c, d, f** or **l** for block special file, character special file, directory, plain file, or symbolic link.

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

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

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

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

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

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

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

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

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

−print Always true; the current pathname is printed.

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

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

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).
2) The negation of a primary ('!' is the unary *not* operator).
3) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
4) Alternation of primaries ('**−o**' is the *or* operator).

EXAMPLE

In our local development system, we keep a file called *TIMESTAMP* in all the manual page directories. Here is how to find all entries that have been updated since *TIMESTAMP* was created:

```
angel% find /usr/man/man2 −newer /usr/man/man2/TIMESTAMP −print
/usr/man/man2
/usr/man/man2/socket.2
/usr/man/man2/mmap.2
angel%
```

To find all the files called *intro.mss* starting from the current user's directory:

```
angel% find . −name intro.mss −print
./manuals/assembler/intro.mss
./manuals/sun.core/intro.mss
./manuals/driver.tut/intro.mss
./manuals/users.guide/intro.mss
./manuals/refman/intro.mss
./manuals/sys.manager/intro.mss
./manuals/sys.manager/uucp.impl/intro.mss
./supplements/general.works/unix.introduction/intro.mss
./supplements/programming.tools/sccs/intro.mss
angel%
```

To remove all files named 'a.out' or '*.o' that have not been accessed for a week:

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

Note that the { } characters must be quoted when using the C shell.

FILES

/etc/passwd
/etc/group

SEE ALSO

sh(1), test(1), fs(5)

BUGS

The syntax is painful.

NAME

fmt – simple text formatter

SYNOPSIS

fmt [**-width**] [**-c**] [*filename* ...]

DESCRIPTION

Fmt is a simple text formatter which reads the concatenation of input files (or standard input if none are given) and produces on standard output a version of its input with lines as long as possible without execeeding *width* characters. *Width* defaults to 72. Blank lines and interword spacing is preserved in the output.

OPTIONS

–*width* Fill output lines to within *width* columns.

–c Crown margin mode – the first two lines following an empty line retain their indention. Subsequent non-blank lines are aligned with the second.

In normal mode, the spacing at the beginning of the input lines is preserved in the output. Lines retain their separation if the input indenting increases, but may be concatenated otherwise.

Fmt is meant to format mail messages prior to sending, but may also be useful for other simple tasks. For instance, in the *vi* text editor, the command:

!}fmt

reformats a paragraph, making the lines reasonably even length.

SEE ALSO

nroff(1), mail(1)

BUGS

Fmt was designed to be simple and fast – for more complex operations, the standard text processors are likely to be more appropriate.

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [–*width*] [file ...]

DESCRIPTION

Fold is a filter which folds the contents of the specified *files*, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1) before using *fold*.

SEE ALSO

expand(1)

BUGS

Folding may not work correctly if underlining is present.

NAME

fonttool — a *vfont* screen-font editor

SYNOPSIS

fonttool [**−W**[generic_tool_arg]] [*font_name*]

DESCRIPTION

fonttool is an editor for fixed-width fonts in *vfont*(5) format whose characters are no taller than 24 pixels (larger characters will not fit completely onto the screen).

OPTIONS

−W*generic_tool_arg*

fonttool accepts any generic tool argument as described in *suntools(1)*. Otherwise, you can manipulate the tool via the Tool Manger Menu.

COMMANDS

To edit a font, type '**fonttool**'. A *font_name* may be supplied on the command line or may be typed into the option subwindow once the program has started. If it exists, the *font_name* file must be in *vfont*(5) format. When the program starts, it displays a single large window containing four subwindows. From top to bottom, the four subwindows are:

1) The top subwindow, a message subwindow, displays messages, prompts, and warnings.

2) The second subwindow from the top, an option subwindow, allows you to set global parameters for the entire font and specify operations for editing any single character. The options are:

(Read) Read in the font specified in the file name field. The program will warn you if you try to read over a modified font.

(Save) Write the current font onto disk with the name in file name field.

(Exit) Quit the program; warns you if you have modified the font.

Font name:
The name of the font.

Max Width and **Max Height**:
The size, in pixels, of the largest character in the font. If you edit an existing font, these parameters are set automatically; you must set them if you are creating a new font. Changing either of these values for an existing font may alter the glyph of some characters of the font. If the glyph size of a character is larger than the new max size, then that character is clipped to the new size (its bottom and right edges are moved in). However, if a glyph's size is smaller than the new size, the glyph is left alone.

Caps Height:
The top of a capital letter in the font is this many pixels above the baseline.

X-Height:
The top of a lower case letter (that is, a lower case "x") is this many pixels above the baseline.

Note: When an existing font is edited, the values of **Caps Height** and **X-Height** are estimated by *fonttool*, and may require some adjustment.

Baseline: The number of pixels from the top (*i.e.*, the upper left corner) of the character to the baseline. For an existing font, the value of the largest baseline distance is used. For a new font, each character will have the same baseline distance. If this value is changed, then the baseline distance for all characters in the font will be the new value.

(Apply) Apply the current values of **Max width**, **Max Height**, **Caps Height**, **X-**

Height, and **Baseline** to the font. That is, changes made to these values do not take effect until **Apply** is selected.

Operation:

This is a list of drawing and editing operations that you can perform on a character. For drawing, the left mouse button draws in black, and the middle draws in white. Operations are:

Single Pt Press a mouse button down and a grey cell will appear; move the mouse and the cell will follow it. Releasing the the button will draw.

Pt Wipe Pressing a button down will draw and moving with the button down will continue drawing until the button is released.

Line Button down marks the end point of a line; moving with the button down rubber bands a line; releasing button draws the line.

Rect Like **Line** except draws a rectangle.

Cut Button down marks one end of rectangle, and moving rubber bands the outline of the rectangle. Button up places the contents of the rectangle into a buffer and then "cuts" (draws in white) the rectangular region from the character. The **Paste** operation (below) gets the data from the buffer.

Copy Like **Cut** except that the region is just copied; no change is made to the character.

Paste Button down displays a rectangle the size of the region in the buffer. Moving with the button down moves the rectangle. Button up pastes the contents of the buffer into the character.

Note: The contents of the **paste** buffer cannot be transferred between tools.

In **Copy** or **Cut** mode, holding down the shift key while pressing the left or middle mouse button will preform a "**Paste**" action. For best results, after placing a region in the buffer, press down the shift key and hold it down, then press down the mouse button. Release the mouse key to paste the region and then release the shift key.

3) The third subwindow echoes the characters in the current font as they are typed. Note that the cursor must be in this window in order to see the characters. Your character delete key will delete the echoed characters.

4) The bottom subwindow, the editing subwindow, displays eight smaller squares at its top; these are called **edit buttons**. The top section of each of these buttons contains a line of text in the form *nnn: c*, where *nnn* is the hexidecimal number of the character and *c* is the standard ASCII character corresponding to that number. In the lower section of the button the character of the current font, if it exists, is displayed. Clicking once over an editing button selects its character for editing.

Just below this row of buttons is a box with the characters "0 9 A Z a z" in it. This box is called a **slider** The slider allows you to scroll around in the font and select which section of the font you want displayed in the edit buttons. The black rectangle near "a" is an indicator which shows the section of the font that is displayed in the buttons above. To move the indicator, select it by pressing the left or middle mouse button down over the indicator and then move the mouse to the left or right with the button down; the indicator will slide along with the cursor. Releasing the button selects the new section of the font. A faster method of moving about in the font is to just press down and release the mouse button above the area you want without bothering to drag the indicator. Another method of scrolling through the characters of the font is to press a key on the keyboard when the cursor is in the bottom window; that character is the first one displayed in the edit buttons.

EDITING CHARACTERS:

To edit a character, click once over the edit button where the character is displayed. When you do this, an edit pad will appear in the bottom subwindow.

The edit pad consists of an editing area bordered by scales, a proof area, and 3 command buttons. The editing area is **Max. Width** by the **Max. Height** when the pad opens, and displays a magnified view of the selected character. Black squares indicate foreground pixels. The editing area is surrounded by scales which show the current **Caps Height**, **X-Height** and **Baseline** in reverse video.

Just outside the scales, on the top, right side, and bottom of the pad, are three small boxes with the capital letters "R", "B", and "A" in them. These boxes are movable sliders that change the right edge, bottom edge, and x-axis advance of the character respectively. In a fixed-width font, these values are usually the same for all characters; however, in a variable-width font these controls can be used to set these properties for each character.

To the right of the pad is the proof area where the character is displayed at normal (that is, screen) resolution and three buttons. The three buttons are:

Undo Clicking the left or middle mouse button undoes the last operation.

Save Stores the current representation of the character in the font.

Quit Closes the edit pad.

In the bottom subwindow, the right mouse button displays a menu of operations. These operations are the same as those in the option subwindow discussed above; you can select the current operation by either picking the operation in the option subwindow or by selecting the appropriate menu with the left button of the mouse. When the cursor is in the other subwindows, the left button displays the standard tool menu.

FILES

/usr/lib/fonts/fixedwidthfonts – Sun-supplied screen fonts

SEE ALSO

suntools(1), *vfont*(5), *vswap(1)*

BUGS

Results are unpredictable with variable-width fonts. The baseline should be greater than 0 or else the font cannot be read in by *fonttool* or by *suntools*(1).

NAME

fpr – print Fortran file

SYNOPSIS

fpr

DESCRIPTION

Fpr is a filter that transforms files formatted according to Fortran's carriage control conventions into files formatted according to UNIX line printer conventions.

Fpr copies its input onto its output, replacing the carriage control characters with characters that will produce the intended effects when printed using *lpr*(1). The first character of each line determines the vertical spacing as follows:

(blank)	one line
0	two lines
1	to first line of next page
+	no advance

A blank line (that is, an empty line) is treated as if its first character is a blank. A blank that appears as a carriage control character is deleted. A zero is changed to a newline. A one is changed to a form feed. The effects of a "+" are simulated using backspaces.

Note that *fpr* is known as *asa* in UNIX System V.

EXAMPLES

```
a.out ¦ fpr ¦ lpr
```

```
fpr < f77.output ¦ lpr
```

BUGS

Results are undefined for input lines longer than 170 characters.

NAME

from – who is my mail from?

SYNOPSIS

from [**−s***sender*] [user]

DESCRIPTION

From prints out the mail header lines in your mailbox file to show you who your mail is from. If *user* is specified, then *user*'s mailbox is examined instead of your own.

OPTIONS

−s*sender*

Only display headers for mail sent by *sender*.

FILES

/usr/spool/mail/*

SEE ALSO

biff(1), mail(1), prmail(1)

NAME

fsplit – split a multi-routine Fortran file into individual files

SYNOPSIS

fsplit [**–e** efile] ... [file]

DESCRIPTION

Fsplit takes as input either a file or standard input containing Fortran source code. It attempts to split the input into separate routine files of the form *name.f,* where *name* is the name of the program unit (e.g. function, subroutine, block data or program). The name for unnamed block data subprograms has the form *blkdtaNNN.f* where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form *mainNNN.f.* If there is an error in classifying a program unit, or if *name.f* already exists, the program unit will be put in a file of the form *zzzNNN.f* where *zzzNNN.f* does not already exist.

Normally each subprogram unit is split into a separate file. When the –*e* option is used, only the specified subprogram units are split into separate files. E.g.:

```
fsplit –e readit –e doit prog.f
```

will split readit and doit into separate files.

DIAGNOSTICS

If names specified via the -*e* option are not found, a diagnostic is written to *standard error.*

BUGS

Fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse *fsplit.*

It is hard to use **–e** for unnamed main programs and block data subprograms since you must predict the created file name.

NAME

ftp – file transfer program

SYNOPSIS

ftp [**–v**] [**–d**] [**–i**] [**–n**] [*host*]

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. *Ftp* transfers files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* immediately attempts to establish a connection to an FTP server on that host; otherwise, *ftp* enters its command interpreter and awaits instructions from the user. When *ftp* is awaiting commands from the user, it displays the prompt "ftp>".

OPTIONS

Options may be specified at the command line, or to the command interpreter.

–v show all responses from the remote server, as well as report on data transfer statistics.

–n do not attempt "auto-login" upon initial connection. If auto-login is enabled, *ftp* checks the *.netrc* file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* uses the login name on the local machine as the user identity on the remote machine, and prompts for a password and, optionally, an account with which to login.

–i turn on interactive prompting during mutliple file transfers (unimplemented).

–d enable debugging.

COMMANDS

! Invoke a shell on the local machine.

ascii Set the file transfer *type* to network ASCII. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the file transfer *type* to support binary image transfer.

bye Terminate the FTP session with the remote server and exit *ftp*.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close Terminate the FTP session with the remote server, and return to the command interpreter.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string "-->".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal.

form *format*
Set the file transfer *form* to *format*. The default format is "file".

get *remote-file* [*local-file*]

Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

mkdir *directory-name*
Make a directory on the remote machine.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any mget or mput will transfer all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for bye.

quote*arg1 arg2* ...
The arguments specified are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

recv *remote-file* [*local-file* [
A synonym for get.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

rename [*from* [[*to*]
Rename the file *from* on the remote machine, to the file *to*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for put.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default "stream" structure is used.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. Unless *ftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for help.

Command arguments which have imbedded spaces may be quoted with quote (") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name "" is specifed, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is "¦", the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell, using *popen*(3S) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ""¦ ls -lt"". A particularly useful example of this mechanism is: "dir ¦more".

3) Failing the above checks, local file names are expanded according to the rules used in the *csh*(1). After the name is expaned, if more than one filename matches, only the first filename is used.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of "ascii", "image" (binary), "ebcdic", and "local byte size" (for PDP-10's and PDP-20's mostly). *Ftp* supports the ascii and image types of file transfer.

Ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

BUGS

Many FTP server implementations do not support experimental operations such as print working directory. Aborting a file transfer does not work right; if one attempts this the local *ftp* will likely have to be killed by hand. VAX sites running the BBN FTP server appear to ignore the PORT command while indicating complicity; this locks up all file transfers.

NAME

gcore – get core images of running processes

SYNOPSIS

gcore process-id ...

DESCRIPTION

Gcore creates a core image of each specified process. Such an image can be used with *adb*(1) or *dbx*(1).

FILES

core.<process-id>	core images

BUGS

Paging activity that occurs while *gcore* is running may cause the program to become confused. For best results, the desired processes should be stopped.

NAME

get − get a version of an SCCS file

SYNOPSIS

/usr/sccs/get [**−r SID**] [**−c cutoff**] [**−i list**] [**−x list**] [**−a seq-no.**] [**−k**] [**−e**] [**−l**[**p**]]
[**−p**] [**−m**] [**−n**] [**−s**] [**−b**] [**−g**] [**−t**] file **...**

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specified option. Arguments may be specified in any order, options apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

OPTIONS

Options are explained below as though only one SCCS file is to be processed, but the effects of any option argument applies independently to each named file.

−r *SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the **−e** option is also used), as a function of the SID specified.

−c *cutoff*

Cutoff date-time, in the form: YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: **−c77/2/2 9:22:25**. Note that this implies that one may use the %E% and %U% identification keywords.

−e This *get* is for editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). A **/usr/sccs/get −e** applied to a particular version (SID) of the SCCS file prevents further **/usr/sccs/get −e** commands on the same SID until *delta* is run or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **/usr/sccs/get −e** for different SIDs is always allowed.

If the *g-file* generated by a **/usr/sccs/get −e** is accidentally ruined in the process of editing it, it may be regenerated by re-running a *get* with the **−k** option in place of the **−e** option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **−e** option is used.

−b Used with the **−e** option to indicate that the new delta should have an SID in a new branch as shown in Table 1. This option is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

−i *list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= SID | SID − SID

SID, the SCCS Identification of a delta, may be in any form shown in the 'SID Specified' column of Table 1. Partial SIDs are interpreted as shown in the 'SID Retrieved' column of Table 1.

−x *list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **−i** option for the *list* format.

−k Suppress replacement of identification keywords (see below) in the retrieved text by their value. The **−k** option is implied by the **−e** option.

−l[**p**] Write a delta summary into an *l-file*. If **−lp** is used, the delta summary is written on the standard output and the *l-file* is not created. See *FILES* for the format of the *l-file*.

−p Write the text retrieved from the SCCS file to the standard output. No *g-file* is created. All output which normally goes to the standard output goes to the standard error file instead, unless the **−s** option is used, in which case it disappears.

−s Suppress all output normally written on the standard output. However, fatal error messages (which always go to the standard error file) remain unaffected.

−m Precede each text line retrieved from the SCCS file with the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Precede each generated text line with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** options are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** option generated format.

−g Do not actually retrieve text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Access the most recently created ('top') delta in a given release (for example, **−r1**), or release and level (for example, **−r1.2**).

−a *seq-no.*
The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This option is used by the *comb*(1) command; it is not a generally useful option, and users should not use it. If both the **−r** and **−a** options are specified, the **−a** option is used. Care should be taken when using the **−a** option in conjunction with the **−e** option, as the SID of the delta to be created may not be what one expects. The **−r** option can be used with the **−a** and **−e** options to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** option is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** option is used included deltas are listed following the notation 'Included'; if the **−x** option is used, excluded deltas are listed following the notation 'Excluded'.

TABLE 1. Determination of SCCS Identification String

SID* Specified	—**b** Option Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* 'R', 'L', 'B', and 'S' are the 'release', 'level', 'branch', and 'sequence' components of the SID, respectively; 'm' means 'maximum'. Thus, for example, 'R.mL' means 'the maximum level number within release R'; 'R.L.(mB+1).1' means 'the first sequence number on the *new* branch (that is, maximum branch number plus one) of level L within release R'. Note that if the SID specified is of the form 'R.L', 'R.L.B', or 'R.L.B.S', each of the specified components *must* exist.

** 'hR' is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** Forces creation of the *first* delta in a *new* release.

# Successor.

† The —**b** option is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of — means 'irrelevant'.

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as 'this shouldn't have happened' type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string **@(#)** recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*, These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** option is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** option is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   * if the delta wasn't applied and wasn't ignored.
c. A code indicating a 'special' reason why the delta was or was not applied:
   'I': Included.
   'X': Excluded.
   'C': Cut off (by a **−c** option).
d. Blank.
e. SCCS identification (SID).

f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* passes information resulting from a **/usr/sccs/get —e** along to *delta*. Its contents are also used to prevent a subsequent execution of a **/usr/sccs/get —e** for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **—i** option if it was present, followed by a blank and the **—x** option if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (that is, *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

sccs(1), admin(1), delta(1), help(1), prs(1), what(1), sccsfile(5).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, only one file may be named when the **—e** option is used.

NAME

gprof − display call graph profile data

SYNOPSIS

gprof [**−a**] [**−b**] [**−c**] [**−e** *name*] [**−E** *name*] [**−f** *name*] [**−F** *name*] [**−s**] [**−z**]
[*object-file* [*profile-file* ...]]

DESCRIPTION

Gprof produces an execution profile of C, Pascal, or FORTRAN 77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (*gmon.out* default) which is created by programs compiled with the **−pg** option of *cc*, *pc*, and *f77*. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the named object file (*a.out* default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendants. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendants is propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

The profiled program must call *exit*(2) or return normally for the profiling information to be saved in the *gmon.out* file.

OPTIONS

−a suppress printing statically declared functions. If this option is given, all relevant information about the static function (for instance, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

−b display a description of each field in the profile.

−c the static call graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

−e *name*
suppress printing the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **−e** option may be given. Only one *name* may be given with each **−e** option.

−E *name*
suppress printing the graph profile entry for routine *name* (and its descendants) as **−e**, above, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. More than one **−E** option may be given. For example, **−E** *mcount* **−E** *mcleanup* is the default.

—f *name*
print the graph profile entry only for routine *name* and its descendants. More than one **—f** option may be given. Only one *name* may be given with each **—f** option.

—F *name*
print the graph profile entry only for routine *name* and its descendants (as **—f**, above) and also uses only the times of the printed routines in total time and percentage computations. More than one **—F** option may be given. Only one *name* may be given with each **—F** option. The **—F** option overrides the **—E** option.

—s produce a profile file *gmon.sum* which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of *gprof* (probably also with a **—s**) option to accumulate profile data across several runs of an *a.out* file.

—z display routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **—c** option for discovering which routines were never called.

FILES

a.out	the namelist and text space
gmon.out	dynamic call graph and profile
gmon.sum	summarized dynamic call graph and profile

SEE ALSO

monitor(3), profil(2), cc(1), prof(1)
Graham, S.L., Kessler, P.B., McKusick, M.K., 'gprof: A Call Graph Execution Profiler', *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

NAME

gfxtool – Run graphics programs in the SunWindows environment

SYNOPSIS

gfxtool [**–C**] [*program* [*arguments*]]

OPTIONS

–C Redirect system console output to this instance of *gfxtool.*

gfxtool also accepts all of the generic tool arguments; see *suntools*(1) for a list of these arguments.

If a *program* argument is present, *gfxtool* runs it. If there are no arguments, *gfxtool* runs the program corresponding to your SHELL environment variable. If this environment variable is not available, then *gfxtool* runs */bin/sh.*

DESCRIPTION

gfxtool is a standard tool provided with *SunWindows.* It allows you to run graphics programs that don't overwrite the terminal emulator from which they run.

gfxtool has two subwindows: a terminal subwindow and an empty subwindow. The terminal subwindow contains a running shell, just like the shelltool (see *shelltool*(1)). Programs invoked in the terminal subwindow can run in the empty subwindow. You can move the boundary between these two subwindows as described in *suntools*(1) under *The Tool Manager.* If you wish, you can make *gfxtool* your console by entering a first argument of -C.

Normally you can use the mouse and keyboard anywhere in the empty subwindow to access Tool Manager functions. However, some graphics programs which run in this window may take over inputs directed to it. For example, SunCore uses the mouse and keyboard for its own input. When you run such tools, access the Tool Manager menu from the tool boundaries or namestripe.

Most of the graphics demo programs in /usr/demo will run in *gfxtool.* In particular, the following demos run in *gfxtool:*

/usr/demo/bouncedemo
Displays a bouncing square. Place a **-n** followed by a decimal number on the command line to indicate how many repetitions of the bounce sequence should be done.

/usr/demo/spheresdemo
Laboriously computes a random collection of shaded spheres. Place a **–n** followed by a decimal number on the command line to indicate how many spheres should be drawn. Colored spheres are drawn on color displays.

/usr/demo/jumpdemo
Simulates the famous *Star Wars* jump to light-speed sequence using vector drawing. Place a **–n** followed by a decimal number up to 500 to indicate how many stars should be used in the star field. Colored stars are drawn on color displays. A **-c** on the command line directs the program to rotate the color map, thus producing a sparkling effect.

/usr/demo/framedemo
Displays a series of frames, each of which contains a 256 by 256 image formed of pixels one deep (that is, the image is a square monochrome bitmap, with 256 bits on a side). The frames must be in the files *frame.1* through *frame.n* on the current working directory, and are displayed in numerical order. A set of sample frames is available to you in the directory */usr/demo/globeframes/*.* Put a **–n** followed by a decimal number on the command line to indicate how many times to cycle through the frames. If you move the cursor into the image window, you can type certain characters to affects the rate at which the frames are displayed. The initial rate is one frame per second. Typing "S" causes an additional one second delay between frames. Typing "F" removes one second from the inter-frame delay. Typing "s" adds 1/20th of a second. Typing "f" removes

1/20th of a second. Typing "Ff" makes the delay as small as possible.

For all these demos the following is true: if no **−n** flag appears on the command line then the program runs continuously until you interrupt it. A **−r** flag on the command line turns the window into a *retained* window. This allows the image to reappear when uncovered instead of restarting the demo. You can also invoke all these demos from the workstation console, i.e., outside of the *suntools* window environment. A **−d** flag followed by a *display device* name as in "bouncedemo -d /dev/cgone0" directs the demo to run on a display other than the console.

SEE ALSO

suntools(1)
shelltool(1)

FILES

~/.ttyswrc
/usr/bin/gfxtool
/usr/bin/suntools
/usr/demo/*
/usr/src/sun/suntool/gfxtool.c

BUGS

If more than 256 characters are input to a terminal emulator subwindow without an intervening newline, the terminal emulator may hang. If this occurs, display the Tool Manager Menu; the "TTY Hung?" submenu there has one item, "Flush input", that you can invoke to correct the problem.

NAME

graph – draw a graph

SYNOPSIS

graph [**–a** *spacing* [*start*]] [**–b**] [**–c** *string*] [**–g** *gridstyle*] [**–l** *label* [
[**–m** *connectmode*] [**–s**] [**–x** [**l**] *lower* [*upper* [*spacing*]]]
[**–y** [**l**] *lower* [*upper* [*spacing*]]] [**–h** *fraction*] [**–w** *fraction*] [**–r** *fraction*]
[**–u** *fraction*] [**–t**] **...**

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *plot*(1G) filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes **"..."**, in which case they may be empty or contain blanks and numbers; labels never contain newlines.

A legend indicating grid range is produced with a grid unless the **–s** option is present.

OPTIONS

Each option is recognized as a separate argument.

–a *spacing* [*start*]
Supply abscissas automatically (they are missing from the input); *spacing* is the spacing (default 1). *start* is the starting point for automatic abscissas-(default 0 or lower limit given by **–x**).

–b Break (disconnect) the graph after each label in the input.

–c *string*
String is the default label for each point.

–g *gridstyle*
Gridstyle is the grid style: 0 no grid, 1 frame with ticks, 2 full grid (default).

–l *label* is label for graph.

–m *connectmode*
is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

–s Save screen, don't erase before plotting.

–x [**l**] *lower* [*upper* [*spacing*]]
If **l** is present, *x* axis is logarithmic. *lower*and*upper* are lower (and upper) *x* limits. *spacing*, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

–y [**l**] *lower* [*upper* [*spacing*]]
If **l** is present, *y* axis is logarithmic. *lower*and*upper* are lower (and upper) *y* limits. *spacing*, if present, is grid spacing on *y* axis. Normally these quantities are determined automatically.

–h *fraction*
is fraction of space for height.

–w *fraction*
is fraction of space for width.

–r *fraction*
is fraction of space to move right before plotting.

–u *fraction*

is fraction of space to move up before plotting.

−t Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1G), plot(1G)

BUGS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

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

SYNOPSIS

grep [**−v**] [**−c**] [**−l**] [**−n**] [**−b**] [**−l**] [**−s**] [**−h**] [**−w**] [**−e** *expression*] expression
[filename ...]

egrep [**−v**] [**−c**] [**−l**] [**−n**] [**−b**] [**−s**] [**−h**] [**−e** *expression*] [**−f** *file*] [expression]
[filename] ...]

fgrep [**−v**] [**−x**] [**−c**] [**−l**] [**−n**] [**−b**] [**−i**] [**−s**] [**−h**] [**−e** *expression*] [**−f** *file*]
[strings] [filename] ...]

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). *Egrep* patterns are full regular expressions including alternation. *Fgrep* searches for lines that contain one of the (newline-separated) *strings*. *Fgrep* patterns are fixed strings — no regular expression metacharacters are supported.

In general, *egrep* is the fastest of these programs.

Take care when using the characters $ * [^ | ( ) and \ in the *expression* as these characters are also meaningful to the Shell. Enclose the entire *expression* argument in single quotes ´ ´ if you need any of the above special characters in the *expression*.

When any of the *grep* utilities is applied to more than one input file, the name of the file is displayed preceding each line which matches the pattern. The filename is not displayed when processing a single file, so if you actually want the filename to appear, use */dev/null* as a second file in the list.

OPTIONS

−v Invert the search to only display lines that *do not* match.

−x Display only those lines which match exactly — that is, only lines which match in their entirety (*fgrep* only).

−c Display a count of matching lines.

−l List the names of files with matching lines (once) separated by newlines.

−n Precede each line by its relative line number in the file.

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

−i Ignore the case of letters in making comparisons — that is, upper and lower case are considered identical. This applies to *grep* and *fgrep* only.

−s Work silently, that is, display nothing except error messages. This is useful for checking the error status.

−h Do not display filenames.

−w search for the expression as a word as if surrounded by '\<' and '\>', see *ex*(1). *grep* only.

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

−f *file* Take the regular expression (*egrep*) or string list (*fgrep*) from *file*.

REGULAR EXPRESSIONS

In the following description 'character' excludes newline:

**** Is an escape character: \ followed by any single character other than newline matches that character.

^ Anchored match: matches the beginning of a line.

$ Anchored match: matches the end of a line.

. (period) matches any character.

c Where *c* is any single character not otherwise endowed with special meaning matches that character.

[*string*]
Character class: match any single character from *string*. Ranges of ASCII character codes may be abbreviated as in 'a–z0–9'. A] may occur only as the first character of the string. A literal – must be placed where it can't be mistaken as a range indicator. A ^ (circumflex) character immediately after the open bracket negates the sense of the character class, that is, the pattern matches any character *except* those in the character class.

* Closure: a regular expression followed by an * (asterisk) matches a sequence of 0 or more matches of the regular expression.

+ Closure: a regular expression followed by a + (plus) matches a sequence of 1 or more matches of the regular expression.

? Closure: a regular expression followed by a ? (question mark) matches a sequence of 0 or 1 matches of the regular expression.

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

¦ Alternation: two regular expressions separated by ¦ or newline match either a match for the first or a match for the second (*egrep* only).

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

The order of precedence of operators at the same parenthesis level is [ ] (character classes), then * + ? (closures), then concatenation, then ¦ (alternation) and newline.

EXAMPLES

Search a file for a fixed string using *fgrep*:

```
tutorial% fgrep intro /usr/man/man3/*.3*
```

Look for character classes using *grep*:

```
tutorial% grep '[1-8]([CJMSNX])' /usr/man/man1/*.1
```

Look for alternative patterns using *egrep*:

```
tutorial% egrep '(Sally¦Fred) (Smith¦Jones¦Parker)' telephone.list
```

To get the filename displayed when only processing a single file, use */dev/null* as the second file in the list:

```
tutorial% grep 'Sally Parker' telephone.list /dev/null
```

SEE ALSO

vi(1)	visual display-oriented editor based on ex(1)
ex(1)	line-oriented text editor based on ed(1)
ed(1)	primitive line-oriented text editor
sed(1)	stream editor
awk(1)	pattern scanning and text processing language
sh(1)	Bourne Shell

DIAGNOSTICS

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

BUGS

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

Ideally there should be only one *grep,* but for historical reasons there are three different versions each with a slightly different set of options and syntaxes.

NAME

groups – show group memberships

SYNOPSIS

groups

DESCRIPTION

Groups displays the groups to which you belong. Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group*. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

When a new file is created it is given the group of the containing directory.

SEE ALSO

setgroups(2)

FILES

/etc/passwd, /etc/group

NAME

head — display first few lines of specified files

SYNOPSIS

head [−count] [file **...**]

DESCRIPTION

Head copies the first *count* lines of the specified file(s), or of the standard input if no filename is given, to the standard output. The default value of *count* is 10 lines.

When more than one file is specified, *head* places a marker at the start of each file which looks like:

==> *filename* <==

Thus, a common way to display a set of short files, identifying each one, is:

```
gaia% head -9999 file1 file2 . . .
```

EXAMPLE

```
gaia% head -4 /usr/man/man1/{cat,head,tail}.1
==> /usr/man/man1/cat.1 <==
.TH CAT 1 "2 June 1983"
.SH NAME
cat — concatenate and display
.SH SYNOPSIS

==> /usr/man/man1/head.1 <==
.TH HEAD 1 "24 August 1983"
.SH NAME
head — display first few lines of specified files
.SH SYNOPSIS

==> /usr/man/man1/tail.1 <==
.TH TAIL 1  "27 April 1983"
.SH NAME
tail — display the last part of a file
.SH SYNOPSIS
```

SEE ALSO

more(1), tail(1), cat(1)

NAME

help – ask for help

SYNOPSIS

/usr/sccs/help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1	Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **ge6**, for message 6 from the *get* command).
type 2	Does not contain numerics (as a command, such as **get**)
type 3	Is all numeric (for example, **212**)

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

When all else fails, try **/usr/sccs/help stuck**.

FILES

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

hostid – print identifier of current host system

SYNOPSIS

hostid

DESCRIPTION

The *hostid* command prints the identifier of the current host in hex. This numeric value is unique across all *Sun* hosts.

SEE ALSO

gethostid(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

icontool – create and edit icons and cursors

SYNOPSIS

icontool [*filename*]

OPTIONS

icontool accepts the SunWindows generic tool arguments; see *suntools*(1) for a list of these arguments.

DESCRIPTION

icontool is a bitmap editor that allows you to create small images such as icons and cursors. *icontool* runs in the SunWindows environment and can be positioned and manipulated in the same way as any other SunWindows tool. For general hints on using SunWindows tools, see *suntools*(1).

When invoked, *icontool* displays a window consisting of several subwindows:

- A large drawing area or *canvas* (on the left).
- A small proof area for previewing a life-size version of the image being edited (at the lower right).
- A control panel showing the options available and their current state.
- An area for status messages (at the upper right).

Instructions for the use of the mouse appear above the canvas.

Move the cursor to the canvas, press and hold the left mouse button, and move the mouse as desired to draw an image. As you draw, an enlarged version of the image appears in the canvas, while a life-sized version of the image appears in the preview area. To "erase" any unwanted portions of the image, press and hold the center mouse button while moving the mouse as desired. Clicking the right mouse button inside the canvas will undo the previous operation.

If you are editing a cursor image, move the cursor into the proof area to turn the proof image into the mouse cursor. This effect only lasts while you are in the proof area; it allows you to test how the new cursor looks during "real" use.

Note that in this writeup, the term "cursor" refers to the mouse cursor, while the term "caret" indicates the current type-in position.

CONTROL PANEL

The control panel at the center right of the tool window lists the options available to you.

To select an option, position the cursor on the desired item and click the left mouse button.

To display a menu for an item, position the cursor on the desired item, then press and hold the right mouse button. To select an alternative from the menu, point at the desired alternative and release the mouse button. If you are just starting out, you may want to use the menus until you become familiar with the tool.

Some of the panel options are *buttons* which you press, in effect, by pointing to them and clicking the left mouse button. Load and Store are two examples of buttons. A small capsule-shaped box encloses each button.

Other choices give you a set of alternatives from which to select. You can cycle through a set of alternatives for a given item by pointing to the item label and repeatedly clicking the left mouse button.

The "painting hand" indicates the current painting mode. Small triangular carets indicate the current state of the Size and grid options as well as the proof background pattern. A triangular caret points to the current type-in position; this can be either the File field or the Fill field corresponding to the Text ("abc") painting mode (see below).

In detail, the control panel options are:

File — Contains the name of the file where the current image resides. To use a different file from the one shown, point at the file name, click the left mouse button, and type a new file name. The filename defaults to the filename given on the command line.

Entering an alphabetic string terminated by <ESC> requests filename completion. *icontool* searches the current directory for files whose names begin with the string you entered. If the filename search locates only one file, that file will be loaded in. In addition, typing CTRL-L, CTRL-S, or CTRL-Q are equivalent to pressing the Load, Store, or Quit buttons, respectively.

Load — (Button) Load the canvas from the file named in the File field.

Store — (Button) Store the current image in the file named in the File field.

Quit — (Button) Terminate processing. Quitting requires a confirming click of the left mouse button.

Size — Alter the canvas size. Choices are icon size (64 x 64 pixels) or cursor size (16 x 16 pixels). Default is icon size.

Grid — Display a grid over the drawing canvas, or turn the grid off. Default is "grid off".

Clear — (Button) Clear the canvas.

Fill — (Button) Fill canvas with current rectangular fill pattern.

Invert — (Button) Invert each pixel represented on the canvas.

Paintbrush

Select from among five modes of painting. The painting hand indicates the current painting mode. Point at the desired mode and click the left mouse button to move the painting hand. The current painting mode remains until you change it. Instructions for each painting mode appear above the canvas. The painting modes are:

dot — Paint a single dot at a time.

line — Draw a line. To draw a line on the canvas, point to the first endpoint of the line, and press and hold the left mouse button. While holding the button down, drag the cursor to the second endpoint of the line. Release the mouse button.

rectangle

Draw a rectangle. To draw a rectangle on the canvas, point to the first corner of the rectangle and press and hold the left mouse button. While holding the button down, drag the cursor to the diagonally opposite corner of the rectangle. Release the mouse button.

In the control panel, the Fill field to the right of the rectangle indicates the current rectangle fill pattern. Any rectangles you paint on the canvas will be filled with this pattern.

circle — Draw a circle. To draw a circle on the canvas, point to the center of the circle, and press and hold the left mouse button. While holding the button down, drag the cursor to the desired edge of the circle. Release the mouse button.

In the control panel, the Fill field to the right of the circle indicates the current circle fill pattern. Any circles you paint on the canvas will be filled with this pattern.

abc — Insert text. To insert text, move the painting hand to "abc" and type the desired text. Then move the cursor to the canvas and press and hold the left mouse button. A box will appear where the text is to go. Position the box as desired and release the mouse button.

In addition, you can choose the font in which to draw the text. Point at the Fill field to the right of the "abc" and either click the left mouse button to cycle through the fonts available (watch the "abc" change as you do this), or press and hold the right mouse button to summon up a menu of fonts.

Load This is the rasterop to be used when loading a file in from disk. See the *SunWindows Reference Manual* for more details on rasterops.

Fill This is the rasterop to be used when filling the canvas. The source for this operation is the rectangle fill pattern, and the destination is the canvas.

Proof This is the rasterop to be used when rendering the proof image. The source for this operation is the proof image, and the destination is the proof background.

Proof background
The proof background can be changed to allow you to preview how the image will appear against a variety of patterns. The squares just above the proof area show the patterns available for use as the proof background pattern. To change the proof background, point at the desired pattern and click the left mouse button.

SEE ALSO

suntools(1)

FILES

/usr/bin/icontool

NAME

indent − indent and format C program source

SYNOPSIS

indent *input-file* [*output-file*] [**−l***nnn*] [**−c***nnn*] [**−cd***nnn*] [**−i***nnn*] [**−dj** ¦ **−ndj**]
[**−v** ¦ **−nv**] [**−bc** ¦ **−nbc**] [**−d***nnn*] [**−br** ¦ **−bl**]

DESCRIPTION

Indent is intended primarily as a C program formatter. Specifically, *indent*:

- indents code lines
- aligns comments
- inserts spaces around operators where necessary
- breaks up declaration lists as in 'int a,b,c;'.

Indent does not break up long statements to make them fit within the maximum line length, but does flag lines that are too long. Lines are broken so that each statement starts a new line, and braces appear alone on a line — see the −br option to inhibit this. Also, an attempt is made to line up identifiers in declarations.

The *flags* which can be specified follow. They may appear before or after the file names.

NOTE: If you specify only an *input-file*, the formatting is done 'in-place', that is, **the formatted file is written back into** *input-file* and a 'backup' copy of *input-file* is written in the current directory. If *input-file* is named '/usr/src/file', the backup file is named *.Bfile*.

If *output-file* is specified, *indent* checks to make sure it is different from *input-file*.

OPTIONS

The options listed here control the formatting style imposed by *indent*.

−l*nnn* Maximum length of an output line. The default is 75.

−c*nnn* The column in which comments start. The default is 33.

−cd*nnn* The column in which comments on declarations start. The default is for these comments to start in the same column as other comments.

−i*nnn* The number of spaces for one indentation level. The default is 4.

−dj,−ndj **−dj** left justifies declarations. **−ndj** indents declarations the same as code. The default is **−ndj**.

−v,−nv **−v** turns on 'verbose' mode, **−nv** turns it off. When in verbose mode, *indent* reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is **−nv**.

−bc,−nbc If **−bc** is specified, then a newline is forced after each comma in a declaration. **−nbc** turns off this option. The default is **−bc**.

−d*nnn* This option controls the placement of comments which are not to the right of code. Specifying **−d2** means that such comments are placed two indentation levels to the left of code. The default **−d0** lines up these comments with the code. See the section on comment indentation below.

−br,−bl Specifying **−bl** lines up complex statements like this:

```
if (...)
{
    code
}
```

Specifying **−br** (the default) makes them look like this:

```
if (...) {
    code
}
```

FURTHER DESCRIPTION

You may set up your own 'profile' of defaults to *indent* by creating a file called *.indent.pro* in your login directory and including whatever switches you like. If *indent* is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, always override profile switches. The profile file must be a single line of not more than 127 characters. The switches should be separated on the line by spaces or tabs.

Multi-line expressions

Indent does not break up complicated expressions that extend over multiple lines, but it usually correctly indents such expressions which have already been broken up. Such an expression might end up looking like this:

```
x =
        (
            (Arbitrary parenthesized expression)
            +
            (
                (Parenthesized expression)
                *
                (Parenthesized expression)
            )
        );
```

Comments

Indent recognizes four kinds of comments. They are: straight text, 'box' comments, UNIX-style comments, and comments that should be passed through unchanged. The action taken with these various types are as follows:

'Box' comments. *Indent* assumes that any comment with a dash immediately after the start of comment (that is, '/*−') is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that the first non-blank character of each successive line is lined up with the beginning slash of the first line. Box comments are be indented (see below).

'Unix-style' comments. This is the type of section header which is used extensively in the UNIX system source. If the start of comment ('/*') appears on a line by itself, indent assumes that it is a UNIX-style comment. These are treated similarly to box comments, except the first non-blank character on each line is lined up with the '*' of the '/*'.

Unchanged comments. Any comment which starts in column 1 is left completely unchanged. This is intended primarily for documentation header pages. The check for unchanged comments is made before the check for UNIX-style comments.

Straight text. All other comments are treated as straight text. *Indent* fits as many words (separated by blanks, tabs, or newlines) on a line as possible. Straight text comments are indented.

Comment indentation

Box, UNIX-style, and straight text comments may be indented. If a comment is on a line with code it is started in the 'comment column', which is set by the **−c***nnn* command line parameter. Otherwise, the comment is started at *nnn* indentation levels less than where code is currently being placed, where *nnn* is specified by the **−d***nnn* command line parameter. Indented comments are never placed in column 1. If the code on a line extends past the comment column, the comment is moved to the next line.

DIAGNOSTICS

Diagnostic error messages, mostly to tell that a text line has been broken or is too long for the output line.

FILES

.indent.pro	profile file

BUGS

Does not know how to format 'long' declarations.

NAME

indxbib – make inverted index to a bibliography

SYNOPSIS

indxbib database ...

DESCRIPTION

Indxbib makes an inverted index to the named *databases* (or files) for use by *lookbib*(1) and *refer*(1). These files contain bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with "%".

Indxbib is a shell script that calls two programs: */usr/lib/refer/mkey* and */usr/lib/refer/inv*. *mkey* truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed — see *Refer — a Bibliography System* in the *Editing and Text Processing on the Sun Workstation* document. *inv* creates an entry file (.ia), a posting file (.ib), and a tag file (.ic), all in the working directory.

FILES

x.ia, *x*.ib, *x*.ic, where *x* is the first argument, or if these are not present, then *x*.ig, *x*

SEE ALSO

refer(1), addbib(1), sortbib(1), roffbib(1), lookbib(1)

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

inews — submit news articles

SYNOPSIS

inews [**−h**] **−t** *title* [**−n** *newsgroups*] [**−e** *expiration date*]

inews −p [*filename*]

inews −C *newsgroup*

DESCRIPTION

Inews submits news articles to the USENET news network. *Inews* is intended as a raw interface to the news system, not as a human user interface. Casual users should probably use *postnews*(1) instead.

The first form of *inews* is for submitting user articles. The body of the article is read from the standard input. A *title* must be specified as there is no default. Each article belongs to a list of newsgroups. The standard list of newsgroups is used if the list is not specified via the **−n** option. On the Sun system, the standard subscription list is: *general, all.general, general, all.announce, !junk, !control,* and *!test.* If you wish to submit an article in multiple newsgroups, the *newsgroups* must be separated by commas and/or spaces. The expiration date is set to the local default if not otherwise specified.

When posting an article, the environment is checked for information about the sender. If NAME is found, its value is used for the full name, rather than the system value obtained from */etc/passwd.* This is useful if the system value cannot be set, or when more than one person uses the same login. If ORGANIZATION is found, the value overrides the system default organization. This is useful when a person uses a guest login and is not primarily associated with the organization owning the machine.

The second form of *inews* is for receiving articles from other machines. If *filename* is given, the article is read from the specified file; otherwise the article is read from the standard input. An expiration date need not be present and a receival date is ignored if present.

After local installation, *inews* transmits the article to all systems that subscribe to the newsgroups that the article belongs to.

The third form of *inews* is for creating new newsgroups. On some systems, this may be limited to specific users such as the super-user or news administrator. This is true on the Sun system.

If the file */usr/lib/news/recording* is present, it is taken as a list of 'recordings' to be shown to users posting news. This is an analogy to the recording you hear when you dial information in some parts of the country, asking you if you really wanted to do this. */usr/lib/news/recording* contains lines of the form:

```
        newsgroups <tab> filename
```

for example:

```
        net.all  net.recording          fa.all   fa.recording
```

Any user posting an article to a newsgroup matching the pattern on the left is shown the contents of the file on the right. The file is found in the LIB directory (often */usr/lib/news*). The user is then told to hit DEL to abort or RETURN to proceed. The intent of this feature is to help companies keep proprietary information from accidently leaking out.

OPTIONS

−n " *newsgroups*"

specifies a list of newsgroups to which the articles are submitted. Elements in the list must be separated by commas and/or spaces. The expiration date is set to the local default if not otherwise specified.

−f [*sender*]

Specifies the article's sender. Without this flag, the sender defaults to the user's name. If **−f** is specified, the real sender's name is included as a Sender line.

−h Headers are present at the beginning of the article, and these headers should be included with the article header instead of as text. This mechanism can be used to edit headers and supply additional nondefault headers, but not to specify certain information, such as the sender and article ID that *inews* itself generates.

FILES

/usr/spool/news/.sys.nnn	temporary articles
/usr/spool/news/*newsgroups/article_no.*	Articles
/usr/spool/oldnews/	Expired articles
/usr/lib/news/active	List of known newsgroups and highest local article numbers in each.
/usr/lib/news/seq	Sequence number of last article
/usr/lib/news/history	List of all articles ever seen
/usr/lib/news/sys	System subscription list

SEE ALSO

mail(1), binmail(1), getdate(3), news(5), newsrc(5), postnews(1), readnews(1), recnews(1), sendnews(8), uucp(1), uurec(8),
Network News User's Guide in the *Beginner's Guide to the Sun Workstation.*

NAME

install — install files

SYNOPSIS

install [**−c**] [**−m** *mode*] [**−o** *owner*] [**−g** *group*] [**−s**] binary destination

DESCRIPTION

Binary is copied to *destination*. If *destination* already exists, it is removed before *binary* is copied. If the destination is a directory then *binary* is copied into file *destination/binary.*

Install refuses to move a file onto itself.

Note: *install* has no special privileges since it simply uses *cp* to copy files from one place to another. The implications of this are:

- You must have permission to read *binary.*
- You must have permission to copy into *destination.*
- You must have permission to change the modes on the final copy of the file if you want to use the **−m** option to change modes. In addition, if you want to set any modes (such as set-user-id), you must be super-user.
- You must be super-user if you want to use the **−o** option to change ownership.

OPTIONS

−c Copy *binary* instead of moving it. In fact, *install always copies the file, but the* **−c** option is retained for backwards compatibility with old system shell scripts which might otherwise break.

−m *mode*
Specifies a different mode for *binary*: the mode for *destination* is set to 755 by default.

−o *owner*
Set the owner of the *destination* file to *owner. destination* is changed to owner *root* by default.

−g *group*
Set the group ownership of the *destination* file to *group. destination* is changed to group *staff* by default.

−s Strip binary files after it is installed — only applicable to binary files in *a.out*(5) format.

SEE ALSO

chmod(1), cp(1), mv(1), strip(1), chown(8)

BUGS

Should be possible to move multiple files at a time, like *mv*(1) or *cp*(1).

When the destination is a directory, *install* simply appends the entire source file name to the directory name, instead of using the source file name's last component like *mv*(1) or *cp*(1).

NAME

join – relational database operator

SYNOPSIS

join [**–a***n*] [**–e** *string*] [**–j**[1|2] *m*] [**–o** *list*] [**–t***c*] 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 separated by blanks, tabs or newlines. Multiple separators count as one, and leading separators are discarded.

OPTIONS

–a*n* The parameter *n* can be one of the values:

1 produce a line for each unpairable line in *file1*.

2 produce a line for each unpairable line in *file2*.

3 produce a line for each unpairable line in both *file1* and *file2*.

The normal output is also produced.

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

–j[1|2] *m*

Join on the *m*th field of file *n*, where *n* is 1 or 2. If *n* is missing, use the *m*th field in each file. Note that *join* counts fields from 1 instead of 0 like *sort*(1) does.

–o *list* Each output line comprises the fields specifed in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. Note that *join* counts fields from 1 instead of 0 like *sort*(1) does.

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

SEE ALSO

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

BUGS

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

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

NAME

kill – send a signal to a process, or terminate a process

SYNOPSIS

kill [–sig] processid ...
kill –l

DESCRIPTION

Kill sends the TERM (terminate, 15) signal to the specified processes. If a signal name or number preceded by '–' is given as first argument, that signal is sent instead of terminate (see *sigvec*(2)). The signal names are listed by using the **–l** option, and are as given in */usr/include/signal.h*, stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal, so **kill –9 ...** is a sure kill, as the KILL (9) signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (that is, processes resulting from the current login) are signaled (but beware: this works only if you use *sh*(1); not if you use *csh*(1).) The killed processes must belong to the current user unless he is the super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a TERM (terminate) signal by 'kill 1'; see *init*(8). To force *init* to close and open terminals according to what is currently in /etc/ttys use 'kill –HUP 1' (sending a hangup, signal 1).

The shell reports the process number of an asynchronous process started with '&' (run in the background). Process numbers can also be found by using *ps*(1).

Kill is built in to *csh*(1); it allows job specifiers, such as **kill %...**, in place of *kill* arguments. See *csh*(1) for details.

OPTIONS

–l Display a list of signal names.

SEE ALSO

csh(1), ps(1), kill(2), sigvec(2)

BUGS

An option to kill process groups ala *killpg*(2) should be provided; a replacement for **kill 0** for *csh*(1) users should be provided.

NAME

last – indicate last logins of users and teletypes

SYNOPSIS

last [*–number*][**–f** *filename*] [name **...**] [tty **...**]

DESCRIPTION

Last looks back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. *Last* displays the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

Last with no arguments displays a record of all logins and logouts, in reverse order.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) *last* indicates how far the search has progressed so far, and the search continues.

OPTIONS

–number
: limit the number of entries displayed to that specified by *number*.

–f *filename*
: Use *filename* as the name of the accounting file instead of */etc/wtmp*.

FILES

/usr/adm/wtmp login data base
/usr/adm/shutdownlog which records shutdowns and reasons for same

SEE ALSO

utmp(5), ac(8), lastcomm(1)

NAME

lastcomm – show last commands executed in reverse order

SYNOPSIS

lastcomm [command name] ... [user name] ... [terminal name] ...

DESCRIPTION

Lastcomm gives information on previously executed commands. *Lastcomm* with no arguments displays information about all the commands recorded during the current accounting file's lifetime. If called with arguments, *lastcomm* only displays accounting entries with a matching command name, user name, or terminal name.

EXAMPLES

```
tutorial% lastcomm a.out root ttyd0
```

would produce a listing of all the executions of commands named *a.out,* by user *root* while using the terminal *ttyd0.* and

```
tutorial% lastcomm root
```

would produce a listing of all the commands executed by user *root.*

For each process entry, *lastcomm* displays the following items of information:

- The command name under which the process was called.
- One or more flags indicating special information about the process. The flags have the following meanings:

  F The process performed a *fork* but not an *exec.*

  S The process ran as a set-user-id program.

  D The process dumped memory.

  X The process was killed by some signal.
- The name of the user who ran the process.
- The terminal which the user was logged in on at the time (if applicable).
- The amount of CPU time used by the process (in seconds).
- The date and time the process exited.

FILES

/usr/adm/acct accounting file

SEE ALSO

last(1), sigvec(2), acct(5), core(5)

NAME

ld – link editor

SYNOPSIS

ld [–A *name*] [**–D** *hex*] [**–d**] [**–e** *entry*] [**–lx**] [**–M**] [**–N**] [**–n**] [**–o** *name*] [**–r**]
[**–S**] [**–s**] [**–Ttext** *hex*] [**–Tdata** *hex*] [**–t**] [**–u** *name*] [**–X**] [**–x**]
[**–y***sym*] [**–z**] *filename* . . .

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *filename*s are given, and *ld* combines them, producing an object module which can either be executed or become the input for a subsequent *ld* run. In the latter case, the –r option must be given to preserve the relocation bits. The output of *ld* is left on a file called *a.out* if not otherwise specified. The output file is made executable only if no errors occurred during link editing.

The argument *filename*s are concatenated in the order specified. The entry point of the output is the beginning of the first routine, unless the –e option is specified.

If a named file is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the same library, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. The first member of a library should be a file named '__.SYMDEF', which is understood to be a dictionary for the library as produced by *ranlib*(1); the dictionary is searched iteratively to satisfy as many references as possible.

The symbols _etext, _edata and _end (**etext, edata and end** in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

OPTIONS

Options should appear before the *filenames*, except abbreviated library names specified by the –l option, which can appear anywhere.

–A *name*

Incremental loading: linking is to be done in a manner so that the resulting object may be read into an already executing program. *Name* is the name of a file whose symbol table is taken as a basis on which to define additional symbols. Only newly linked material is entered into the text and data portions of *a.out*, but the new symbol table will reflect all symbols defined before and after the incremental load. This argument must appear before any other object file in the argument list. One or both of the –T option may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding addresses (which must be a multiple of the page size). The default value is the old value of _end.

–D *hex* Pad the data segment with zero bytes to make it *hex* bytes long.

–d Force definition of common storage even if the

–e *entry*

Define the entry point: the *entry* argument is made the name of the entry point of the loaded program.

–lx This option is an abbreviation for the library name '/lib/lib*x*.a', where *x* is a string. If that does not exist, *ld* tries '/usr/lib/lib*x*.a' A library is searched when its name is encountered, so the placement of a –l is significant.

–M Produce a primitive load map, listing the names of the files which will be loaded.

–N Do not make the text portion read-only or sharable. (Use 'magic number' 0407.)

–n Arrange (by giving the output file a 0410 'magic number') that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible segment boundary following the end of the text.

−o *name*
Name is made the name of the *ld* output file, instead of **a.out**.

−r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics. **−r** flag is present.

−S Strip the output by removing all symbols except locals and globals.

−s Strip the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

−Ttext *hex*
Start the text segment origin at location *hex*. Specifying a bare **−T** is the same as using the **−Ttext** option.

−Tdata *hex*
Start the text segment origin at location *hex*. This option is only of use to programmers wishing to write code for PROM 's, since the resulting code cannot be executed by the UNIX system.

−t Trace: display the name of each file as it is processed.

−u *name*
Enter *name* as an undefined symbol. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−X Record local symbols except for those whose names begin with 'L'. This option is used by *cc*(1) to discard internally-generated labels while retaining symbols local to routines.

−x Do not preserve local (non- **.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

−y*sym* Display each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.

−z Arrange for the process to be loaded on demand from the resulting executable file (0413 'magic number') rather than preloaded. This is the default. Results in a page-sized header on the output file followed by a text and data segment, each of which has a multiple of page-size bytes (being padded out with nulls in the file if necessary). With this format the first few BSS segment symbols may actually end up in the data segment; this is to avoid wasting the space resulting from data segment size roundup.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
/usr/local/lib/lib*.a	still more libraries
a.out	output file

SEE ALSO

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

BUGS

There is no way to force data to be page-aligned.

NAME

leave — remind you when you have to leave

SYNOPSIS

leave [[+]hhmm]

DESCRIPTION

Leave sets an alarm to a time you specify and will tell you when the time is up. *Leave* waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. *Leave* disappears after you log off.

You can specify the time in on of two ways, namely as an absolute time of day in the form *hhmm* where *hh* is a time in hours (on a 12 or 24 hour clock), or you can place a **+** sign in front of the time, in which case the time is relative to the current time, that is, the specified number of hours and minutes from now. All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, *leave* prompts with "When do you have to leave?". *Leave* exits if you just type a newline, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a *.login* or *.profile.*

Leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use **kill −9** and its process id.

SEE ALSO

calendar(1)

EXAMPLES

The first example sets the alarm to an absolute time of day:

```
tutorial% leave 1535
Alarm set for Wed Mar  7 15:35:07 1984

                    work  work  work  work

tutorial% Time to leave!
```

The second example sets the alarm for 10 minutes in the future:

```
tutorial% leave +10
Alarm set for Wed Mar  7 15:45:24 1984

                    work  work  work  work

tutorial% Time to leave!

                    work  work  work  work

tutorial% You're going to be late!
```

BUGS

Relative time does not work as described. Currently you must specify relative time in the form +mmmm. It is not checked against a 12 hour clock.

NAME

lex – generator of lexical analysis programs

SYNOPSIS

lex [**–tvfn**] [file] **...**

DESCRIPTION

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

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

```
cc lex.yy.c –ll
```

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

OPTIONS

–t Place the result on the standard output instead of in file *lex.yy.c*.

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

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

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

EXAMPLE

```
lex lexcommands
```

would draw *lex* instructions from the file *lexcommands,* and place the output in *lex.yy.c*

```
%%
[A–Z] putchar(yytext[0]+´a´–´A´);
[ ]+$
[ ]+   putchar(´ ´);
```

is an example of *lex*. This program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

SEE ALSO

yacc(1), sed(1)
The paper, *LEX – A Lexical Analyzer Generator,* in the Sun *Programming Tools* Manual.

NAME

lint − a C program verifier

SYNOPSIS

lint [**−abchnuvxz**] [**−D** *name=def*] [**−D** *name*] [**−U** *name*] [**−I** *dir*] file ...
lint [**−C***lib*] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* that are likely to be bugs, non-portable, or wasteful. *Lint* also checks the type usage of the program more strictly than the C compiler. *Lint* runs the C preprocessor as its first phase.

Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as **−lm**, in the style of *ld*(1). The standard C library (**−lc**) is *lint*'ed by default. Arguments ending in *.ln* are also treated as library files.

To create lint libraries, use the **−C** option. For example

```
tutorial% lint −Ccongress files . . .
```

where *files* are the C sources of library *congress*, produces a file *llib-lcongress.ln* in the */usr/lib/lint* directory in the correct library format suitable for *lint*'ing programs using **−lcongress**.

OPTIONS

a Report assignments of **long** values to **int** variables.

b Report *break* statements that cannot be reached. This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.

c Complain about casts which have questionable portability.

h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

n Do not check compatibility against the standard library.

u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).

v Suppress complaints about unused arguments in functions.

x Report variables referred to by **extern** declarations, but never used.

z Do not complain about structures that are never defined (for example, using a structure pointer without knowing its contents.).

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

−U*name*
Remove any initial definition of *name* in the preprocessor.

−I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **−I** options, then in the */usr/include* directory.

—C*lib* create a *lint* library with name *lib* (see **DESCRIPTION** section).

—l*lib* use *lint* library *lib* from the */usr/lib/lint* directory.

General Comments

The routine *exit*(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*VARARGS*n**/
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be **0**.

/*ARGSUSED*/
turns on the **—v** option for the next function.

/*LINTLIBRARY*/
at the beginning of a file, shuts off complaints about unused functions in this file.

EXAMPLE

The following *lint* call:

```
tutorial% lint —b myfile
```

checks the consistency of the file 'myfile'. The **—b** option indicates that unreachable **break** statements are to be checked.

FILES

/usr/lib/lint/lint[12]	programs
/usr/lib/lint/llib-lc.ln	declarations for standard functions
/usr/lib/lint/llib-lc	human-readable version of above
/usr/lib/lint/llib-lm.ln	declarations for math functions
/usr/lib/lint/llib-lm	human-readable version of above
/usr/lib/lint/llib-lmp.ln	declarations for multiprecision functions
/usr/lib/lint/llib-lmp	human-readable version of above
lib-l*.ln	library created by the **—C** option

SEE ALSO

cc(1)
Lint, a C Program Checker, in *Programming Tools for the Sun Workstation*

BUGS

There are some things you just *can't* get *lint* to shut up about.

NAME

lockscreen – maintain window context until "login"

SYNOPSIS

lockscreen [**−e**] [**−n**] [**−r**]

DESCRIPTION

Lockscreen preserves the user's window setup and context when the machine is not in use. When run, the dark *lockscreen* display and constantly moving Sun Microsystems logo limit phosphorus burn of the video display that might otherwise occur from running the same window configuration for a long time.

Lockscreen is executed from a terminal emulator running inside the SunWindows system. When any keyboard or mouse button is pressed, the dark screen is replaced by an option panel that displays the user name, a dark box, and a prompt for the user's password. If the user has no password, or if the **−n** option is used, the user's window setup is immediately restored.

When the option screen appears:

1) Enter the user's password to return to the Sunwindows environment; this password is not echoed on the screen, or,

2) Point to the black box and click the left button to return to the dark display.

The panel also allows you to enter a different user name by pointing to the **Name:** field, clicking the left button, and typing in the name. You must then supply the appropriate password.

OPTIONS

−e Add the *Exit Desktop* choice to the options panel. If pointed to and clicked, the Sunwindows environment is exited and the current user is logged out.

−n No password is required to reenter the window environment.

−r Allows use of the user name **root**. Normally, **root** is not accepted as a valid user name.

SEE ALSO

suntools(1), login(1)

NAME

login – sign on

SYNOPSIS

login [username]

DESCRIPTION

Login signs *username* on to the system initially; *login* may also be used at any time to change from one user i.d. to another.

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

After a successful login, accounting files are updated, the user is informed of the existence of mail, and the message of the day is printed, as is the time he last logged in (unless he has a *.hushlogin* file in his home directory — this is mainly used to make life easier for non-human users, such as *uucp*).

Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually either *sh*(1) or *csh*(1)) according to specifications found in a password file. Argument 0 of the command interpreter is "–sh", or more generally the name of the command interpreter with a leading dash ("–") prepended.

Login also initializes the environment *environ*(5) with information specifying home directory, command interpreter, terminal type (if available) and user name.

If the file */etc/nologin* exists, *login* prints its contents on the user's terminal and exits. This is used by *shutdown*(8) to stop logins when the system is about to go down.

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

Login times out and exits if its prompt for input is not answered within some 'reasonable' time.

When the Bourne Shell (*sh*(1)) starts up, it reads a file called *.profile* from the user's home directory. When the C-Shell (*sh*(1)) starts up, it reads a file called *.login* from the user's home directory, and reads a file called *.cshrc* from the user's home directory every time a new C-Shell is started. Note that the Shells only read these files if they are owned by the person who is logging in — these files are not read when *login* is being used to change user i.d.

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/spool/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/nologin	stops logins
.hushlogin	makes login quieter

SEE ALSO

init(8), getty(8), mail(1), passwd(1), passwd(5), environ(5), shutdown(8)

DIAGNOSTICS

"Login incorrect," if the name or the password is bad (or mis-typed).
"No Shell", "cannot open password file", "no directory": ask your system administrator for assistance.

NAME

look – find lines in a sorted list

SYNOPSIS

look [**−df**] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*.

OPTIONS

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

−f Fold: Upper case letters compare equal to lower case.

If no *file* is specified, *look* uses */usr/dict/words* with collating sequence **−df**.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

lookbib — find references in a bibliography

SYNOPSIS

lookbib database

DESCRIPTION

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with "%".

Lookbib uses an inverted index made by *indxbib* to find sets of bibliographic references. It reads keywords typed after the ">" prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another ">" prompt.

It is possible to search multiple databases, as long as they have a common index made by *indxbib*. In that case, only the first argument given to *indxbib* is specified to *lookbib*.

If *lookbib* does not find the index files (the .i[abc] files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a '.ig' suffix, suitable for use with *fgrep*. *Lookbib* then uses this *fgrep* file to find references. This method is simpler to use, but the .ig file is slower to use than the .i[abc] files, and does not allow the use of multiple reference files.

FILES

x.ia, *x*.ib, *x*.ic, where *x* is the first argument, or if these are not present, then *x*.ig, *x*

SEE ALSO

refer(1), addbib(1), sortbib(1), roffbib(1), indxbib(1)

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

lorder — find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

Give *lorder* one or more object or library archive (see *ar*(1)) *files*, and it lists pairs of object file names — meaning that the first file of the pair refers to external identifiers defined in the second — to the standard output. *Lorder*'s output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

EXAMPLE

This brash one-liner intends to build a new library from existing *.o* files.

```
ar cr library ` lorder *.o ¦ tsort`
```

The *ranlib*(1), command converts an ordered archive into a randomly accessed library and makes *lorder* unnecessary.

SEE ALSO

tsort(1), ld(1), ar(1), ranlib(1)

BUGS

The names of object files, in and out of libraries, must end with '.o'; otherwise, nonsense results.

NAME

lpq – spool queue examination program

SYNOPSIS

lpq [**+**[*num*]] [**–l**] [**–P***printer*] [*job #* ...] [*user* ...]

DESCRIPTION

lpq examines the spooling area used by *lpd*(8) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user.

Lpq reports on any jobs currently in the queue when invoked without any options. See the OPTIONS section below for a list of options. Arguments supplied that are not recognized as options are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (that is, invocation of *lpr*(1)) *lpq* reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to *lprm*(1) for removing a specific job), and the total size in bytes. The **–l** option causes information about each of the files comprising the job to be printed. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when *lpr*(1) is used as a sink in a pipeline) in which case the file is indicated as '(standard input)'.

If *lpq* warns that there is no daemon present (that is, due to some malfunction), the *lpc*(8) command can be used to restart the printer daemon.

OPTIONS

Lpq reports on any jobs currently in the queue when invoked without any options.

–P*printer*

report on jobs routed to the specified *printer*. In the absence of the **–P** option, the default line printer is used (or the value of the PRINTER variable in the environment).

+*nnn* display the spool queue until it empties. Supplying a number *nnn* immediately after the **+** sign indicates that *lpq* should sleep *nnn* seconds in between scans of the queue.

FILES

/etc/termcap	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/*	the spooling directory, as determined from printcap
/usr/spool/*/cf*	control files specifying jobs
/usr/spool/*/lock	the lock file to obtain the currently active job

SEE ALSO

lpr(1), lprm(1), lpc(8), lpd(8)

BUGS

The **+** option doesn't wait until the entire queue is empty; it only waits until the local machine's queue is empty.

Due to the dynamic nature of the information in the spooling directory *lpq* may report unreliably.

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

lpq is sometimes unable to open various files because the lock file is malformed.

DIAGNOSTICS

waiting for *printer* to become ready

The daemon could not open the printer device. This can happen for a number of reasons; the most common is that the printer is turned off-line. This message can also be generated if the printer is out of paper, the paper is jammed, and so on. The actual reason is dependent on the meaning of error codes returned by system device driver. Not all printers supply sufficient information to distinguish when a printer is off-line or having trouble (for example, a printer connected through a serial line). Another possible cause of this message is some other process, such as an output filter, has an exclusive open on the device. Your only recourse here is to kill off the offending program(s) and restart the printer with *lpc*.

***printer* is ready and printing**

The *lpq* program checks to see if a daemon process exists for *printer* and prints the file *status*. If the daemon is hung, a super user can use *lpc* to abort the current daemon and start a new one.

waiting for *host* to come up

Indicates that there is a daemon trying to connect to the remote machine named *host* in order to send the files in the local queue. If the remote machine is up, *lpd* on the remote machine is probably dead or hung and should be restarted as mentioned for *lpr*.

sending to *host*

The files should be in the process of being transferred to the remote *host*. If not, the local daemon should be aborted and started with *lpc*.

Warning: *printer* is down

The printer has been marked as being unavailable with *lpc*.

Warning: no daemon present

The *lpd* process overseeing the spooling queue, as indicated in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. The error log file for the printer should be checked for a diagnostic from the deceased process. To restart an *lpd*, use

```
% lpc restart printer
```

NAME

lpr – off line print

SYNOPSIS

lpr [**–P***printer*] [**–#***num*] [**–C***class*] [**–J***job*] [**–T***title*] [**–i**[*num*]] [**–1234***font*] [**–w***num*] [**–r**] [**–m**] [**–h**] [**–s**] [*–filter_option*] [*filename* ...]

DESCRIPTION

Lpr uses a spooling daemon to print the named files when facilities become available. *Lpr* reads the stndard input if no files are specified.

OPTIONS

–P*printer*

Force output to the named *printer*. Normally, the default printer is used (site dependent), or the value of the PRINTER environment variable is used.

–#*num*

Produce multiple copies of output, using *num* as the number of copies for each file named. For example,

```
tutorial% lpr –#3 new.index.c print.index.c more.c
```

produces three copies of the file *new.index.c*, followed by three copies of *print.index.c*, etc. On the other hand,

```
tutorial% cat new.index.c print.index.c more.c | lpr –#3
```

generates three copies of the concatenation of the files.

–C Print *class* as the job classification on the burst page. For example,

```
tutorial% lpr –C Operations new.index.c
```

replaces the system name (the name returned by *hostname*(1)) with 'Operations' on the burst page, and prints the file *new.index.c*.

–J*job* Print *job* as the job name on the burst page. Normally, *lpr* uses the first file's name.

–T*title*

Use *title* instead of the file name for the title used by *pr*(1).

–i[*num*]

Indent output *num* spaces. If *num* is not given, eight spaces are used as default.

–1234*font*

Mount the specified *font* on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size*.

–w*num*

Use *num* as the page width for *pr*(1).

–r Remove the file upon completion of spooling.

–m Send mail upon completion.

–h Suppress printing the burst page.

–s Create a symbolic link from the spool area to the data files rather than trying to copy them (so large files can be printed). This means the data files should not be modified or removed until they have been printed. In the absence of this option, files larger than 1 Megabyte in length are truncated. Note that the **–s** option only works if you are specifically naming data files — it doesn't work if *lpr* is at the end of a pipeline.

filter_option

The following single letter options notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

–p Use *pr*(1) to format the files (equivalent to *print*).

−l Print control characters and suppress page breaks.

−t The files contain data from *troff*(1) (cat phototypesetter commands).

−n The files contain data from *ditroff* (device independent troff).

−d The files contain data from *tex* (DVI format from Stanford).

−g The files contain standard plot data as produced by the *plot*(3X) routines (see also *plot*(1G) for the filters used by the printer spooler).

−v The files contain a raster image for devices like the Versatec.

−c This option currently is unassigned.

−f Interpret the first character of each line as a standard FORTRAN carriage control character.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities data base
/usr/lib/lpd*	line printer daemons
/usr/spool/*	directories used for spooling
/usr/spool/*/cf*	daemon control files
/usr/spool/*/df*	data files specified in "cf" files
/usr/spool/*/tf*	temporary copies of "cf" files

SEE ALSO

lpq(1), lprm(1), pr(1), symlink(2), printcap(5), lpc(8), lpd(8)

DIAGNOSTICS

lpr: copy file is too large

A file is determined to be too 'large' to print by copying into the spool area. Use the **−s** option as defined above to make a symbolic link to the file instead of copying it. A 'large' file is approximately 1 Megabyte in this system.

lpr: *printer***: unknown printer**

The *printer* was not found in the *printcap* database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the */etc/printcap* file.

lpr: *printer***: jobs queued, but cannot start daemon.**

The connection to *lpd* on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket */dev/printer* to be sure it still exists (if it does not exist, there is no *lpd* process running).

lpr: *printer***: printer queue is disabled**

This means the queue was turned off with

tutorial% **lpc disable** *printer*

to prevent *lpr* from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with *lpc*.

If the **−f** and **−s** flags are combined as follows:

lpr −fs *filename*

copies the file to the spooling directory rather than making a symbolic link.

Placing the **−s** flag first, or writing each as separate arguments makes a link as expected.

NAME

lprm − remove jobs from the line printer spooling queue

SYNOPSIS

lprm [**−P***printer*] [**−**] [*job #* ...] [*user* ...]

DESCRIPTION

Lprm removes a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

Lprm without any arguments will delete the currently active job if it is owned by the user who invoked *lprm*.

If the − flag is specified, *lprm* will remove all jobs which a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the *lpr* command was invoked.

Specifying a user's name, or list of user names, will cause *lprm* to attempt to remove any jobs queued belonging to that user (or users). This form of invoking *lprm* is useful only to the super-user.

A user may dequeue an individual job by specifying its job number. This number may be obtained from the *lpq*(1) program. For example:

```
tutorial% lpq −Pimagen
imagen is ready and printing
Rank    Owner   Job   Files                          Total Size
active  wendy   385   standard input                 35501 bytes
tutorial% lprm −Pimagen 305
```

Lprm announces the names of any files it removes and is silent if there are no jobs in the queue which match the request list.

Lprm will kill off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The **−P** option may be used to specify the queue associated with a specific printer (otherwise the default printer, or the value of the PRINTER variable in the environment is used).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently active job

SEE ALSO

lpr(1), lpq(1), lpd(8)

DIAGNOSTICS

lprm: *printer***: cannot restart printer daemon**

The connection to *lpd* on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket */dev/printer* to be sure it still exists (if it does not exist, there is no *lpd* process running). Use

```
% ps ax | fgrep lpd
```

to get a list of process identifiers of running lpd's. The *lpd* to kill is the one which is not listed in any of the "lock" files (the lock file is contained in the spool directory of each printer). Kill the master daemon using the following command.

```
% kill pid
```

Then remove */dev/printer* and restart the daemon (and printer) with the following commands.

```
% rm /dev/prlnter % /usr/llb/lpd
```

Another possibility is that the *lpr* program is not setuid *root*, setgid *spooling*. This can be checked with

```
% ls —lg /usr/ucb/lpr
```

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

ls − list contents of directory

SYNOPSIS

ls [**−acdfgilqrstu1ACLFR**] name ...

DESCRIPTION

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

OPTIONS

There are a large number of options:

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

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

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

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

−g Show the group ownership of the file in a long output.

−i For each file, print the i-number in the first column of the report.

−l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '->'.

−q Display non-graphic characters in file names as the character '?'; this is the default when output is to a terminal.

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

−s Give size in kilobytes of each file.

−t Sort by time modified (latest first) instead of by name.

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

−1 Force one entry per line output format; this is the default when output is not to a terminal.

−A Same as **−a**.

−C Force multi-column output; this is the default when output is to a terminal.

−F Mark directories with a trailing '/', executable files with a trailing '*', symbolic links with a trailing '@', and AF_UNIX domain sockets with a trailing '='.

−L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

−R Recursively list subdirectories encountered.

INTERPRETATION OF LISTING

The mode printed under the **−l** option contains 11 characters interpreted as follows. If the first character is:

- **d** entry is a directory;
- **b** entry is a block-type special file;
- **c** entry is a character-type special file;
- **l** entry is a symbolic link
- **s** entry is an AF_UNIX domain socket, or
- **−** entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

- **r** the file is readable;
- **w** the file is writable;
- **x** the file is executable;
- **−** the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the user-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally '**x**' or '−') is **t** if the 1000 bit of the mode is on. See *chmod*(1) for the meaning of this mode.

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

FILES

/etc/passwd	to get user id's for '*ls* **−l**'.
/etc/group	to get group id's for '*ls* **−g**'.

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as '*ls* **−s**' is much different than '*ls* **−s** ¦ lpr'. On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

NAME

m4 – macro processor

SYNOPSIS

m4 [file **...**]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each argument file is processed in order; the standard input is read if there are no arguments or if an argument is '–'. The processed text is written on the standard output.

Macro calls have the form

```
name(arg1,arg2, . . . , argn)
```

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

Left and right single quotes (` ´) are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string.

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

ifdef If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX versions of *m4*.

changequote
Change quote characters to the first and second arguments. *Changequote* without arguments restores the original values (that is, ` ´).

divert *M4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum returns the value of the current output stream.

dnl reads and discards characters up to and including the next newline.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

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

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

substr returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is identical to *include,* except that it says nothing if the file is inaccessible.

syscmd executes the UNIX command given in the first argument. No value is returned.

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

errprint prints its argument on the diagnostic output file.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.

SEE ALSO

M4 — A Macro Processor , in *Programming Tools for the Sun System.*

NAME

mail – send and receive mail

SYNOPSIS

mail [**–v**] [**–f** [*filename*]] [*people* ...]

INTRODUCTION

Mail is an interactive facility for handling electronic mail. *Mail*'s basic unit of information is a *message*. *Mail* provides facilities for sending messages, browsing through received messages, and sending replies.

Sending mail. To send a message to one or more other *people*, use *mail* with arguments which are the names of *people* to send to. The *people* arguments can be the names of users on your own system, or they can be network addresses — see below. Then, type in your message, followed by an EOT (control–D) at the beginning of a line. The section labelled *Replying to or originating mail,* describes some features of *mail* available to help you compose your letter.

The **–v** option is a debugging aid for *mail* itself: it lets you see if your mail is being sent out OK.

Reading mail. If you use *mail* without arguments, it displays a one-line header for each message. The 'current message' is initially the first message (numbered 1); you can display it with the **print** command (abbreviated **p**). You can move among the messages: use the '+' command to move forward a message, the '–' command to move back a message, and simple numbers to address a particular message.

Disposing of mail. After examining a message you can **delete** (**d**) the message or **reply** (**r**) to it. If you accidentally delete a message, you do have recourse: you can **undelete** (**u**) the message by giving its number, or you can **exit** (**x**) the *mail* session, leaving everything as it was before you started. Messages deleted during the *mail* session however, usually disappear and are never seen again.

Specifying messages. Commands such as **print** and **delete** can be given a list of message-numbers as arguments to apply to a number of messages at once. Numbers in a list are separated by spaces. Message-numbers are simple numbers, or they can be specified as a range, that is, two numbers separated by a minus sign. Thus:

```
        delete 1 2
```

deletes messages 1 and 2, while:

```
        delete 1–5
```

deletes messages 1 through 5. The special name '*' addresses *all* messages, and '$' addresses the *last* message; thus the **top** command (print the first few lines of a message) could be used in:

```
        top *
```

to print the first few lines of all messages.

Replying to mail. You can use the **reply** (**r**) command to respond to a message, followed by the text, and ending with an EOT (control-D) or a period alone on a line. There are some variations of the **reply** command:

r (the straightforward **reply** command) replies to the person who sent you the message. This command can also be typed as **replysender**.

R (upper-case **R**) replies to everyone who got the original message. This command can also be typed as **replyall**.

Also see the **replyall** variable. *Originating mail while browsing.* You can use the *mail* command (**m**) to originate a message while browsing. While you are composing a message, you can use 'escapes' (usually called 'tilde escapes') to get *mail* to treat such lines in special ways. For instance, typing '~m' (alone on a line) places a copy of the current message into the response, but shifted right by one tabstop. Other escapes set up subject fields, add and delete recipients to the message, and allow you to escape to an editor to revise the message or to a shell to run some commands. These options are described in the summary below.

Ending a mail processing session. You end a *mail* session with the **quit** (**q**) command. Messages which have been examined but not deleted go to your *mbox* file. Messages which have been deleted are discarded. Unexamined messages are retained. The **−f** option specifies that *mail* read in the contents of your *mbox* (or the specified file) for browsing; when you **quit** *mail* writes undeleted messages back to this file.

Personal and systemwide distribution lists. You can create personal distribution lists so that, for instance, you can send mail to 'cohorts' and have it go to a group of people. Such lists can be defined by placing a line like:

```
alias cohorts bill ozalp sklower jkf mark cory:kridle
```

in the file *.mailrc* in your home directory. You can nest distribution lists so that a distribution list can refer to other distribution lists, which finally, refer to individual people. The current list of such aliases can be displayed by the **alias** (**a**) command.

System Distribution Lists System wide distribution lists can be created by editing */usr/lib/aliases*, see *aliases*(5) and *sendmail*(8); these are kept in a slightly different syntax. In mail you send, personal aliases are expanded in mail sent to others so that they can **reply** (lower case **r**) to the recipients. System-wide aliases are not expanded when the mail is sent, but any reply returned to the machine will have the system wide alias expanded as all mail goes through *sendmail*. If you edit */usr/lib/aliases*, you must run the program *newaliases*(8) to rebuild the aliases database.

Network mail (ARPA, UUCP). Mail to sites on the ARPA or UUCP network can be sent using *name@site* for ARPA-net sites or *machine!user* for UUCP sites, provided appropriate gateways are known to the system. Be sure to escape the ! in UUCP sites when giving it on a *csh* command line by preceding it with a \.

Mail has a number of options which can be **set** in the *.mailrc* file to alter its behavior; thus

set askcc

enables the 'askcc' (ask for carbon copy) feature. These options are summarized below in the section labelled *Mail Options*.

SUMMARY OF MAIL COMMANDS

This summary is adapted from the *Mail User's Guide*.

Each *mail* command is typed on a line by itself. Arguments may follow the command word. The command word need not be typed in its entirety — the first command which matches the typed prefix is used. Commands which take message lists as arguments use the next message forward which satisfies the command's requirements if a message-list is not specified. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, *mail* types 'No applicable messages' and aborts the command.

−[*nnn*]
: Go to the previous message and print it out. If the optional numeric argument *nnn* is specified, go to the *nnn*'th previous message and print it.

+[*nnn*]
: Go to the next message and print it out. If the optional numeric argument *nnn* is specified, go to the *nnn*'th next message and print it.

? Print a brief summary of commands.

! *Shell Command*
: Execute the UNIX *Shell Command* which follows the !.

alias (**a**) With no arguments, print out all currently-defined aliases. With one argument, print out that alias. With more than one argument, add the users named in the second and later arguments to the alias named in the first argument.

chdir [*directory*]

(c) Change the user's working directory to the optional *directory* argument. Change to the user's login directory if *directory* is not specified.

delete (d) Takes a list of messages as argument and marks them all as deleted. Deleted messages are not saved in *mbox*, nor are they available for most other commands. Messages deleted in the current *mail* session can be undeleted with the **undelete** command described later.

dp (also **dt**) Delete the current message and print the next message. If there is no next message, *mail* says 'at EOF.'

edit (**e**) Takes a list of messages and points the text editor at each one in turn. You can edit the message freely, with two exceptions: you must not touch the first line of the message header (The "From" line), and you must leave a blank line at the end of the message. When you exit from the editor (as normal for the editor: 'write' and 'quit'), the message is read back in.

exit (**ex** or **x**) Effects an immediate return to the Shell without modifying the user's system mailbox, the *mbox* file, or the edit file when using the **−f** option.

folder (**fo**) The **folder** command switches to a new mail file or folder. With no arguments, it tells you which file you are browsing. If you supply a filename as an argument, it will write out changes in the current file, and access the named file. Some special conventions are recognized for the argument. **#** indicates the previous file, **%** indicates your system mailbox, **%***user* indicates the user's system mailbox, **&** means your *~/mbox* file, and **+***folder* means a file in your folder directory.

folders
List the names of the folders in your folders directory.

from (**f**) Takes a list of messages and prints their message headers.

headers
(**h**) List the current range of headers, 20 at a time (fewer lines are displayed on low-speed terminals). If you want to see another group of headers, you must type a number that lies within that group. For example,

headers

alone displays the headers of the first 20 messages. To see headers (say) 41 thru 60, you can type

headers 45

or in fact any number that lies in the range 41 thru 60. Also see the **z** command below, which scrolls forwards and backwards through groups of headers.

help A synonym for ?

hold (**ho**, also **preserve**) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in *mbox*. Does not override the **delete** command.

ignore takes a list of character strings as arguments. Lines in the header of mail messages that start with any of the specified character strings are ignored.

mail (**m**) Takes as arguments login names and distribution group names and sends mail to those people.

next (**n** like **+** or CR) Goes to the next message in sequence and types it. With an argument list, types the next matching message.

preserve
A synonym for **hold.**

print (**p**) Takes a message list and types out each message on the user's terminal. Typing a **print** command with a capital **P** prints even ignored headers.

quit (**q**) Terminate the *mail* session, saving all undeleted, unsaved messages in your *mbox* file in your login directory, preserving all messages marked with **hold** or **preserve** or never referenced in your system mailbox, and removing all other messages from your system mailbox. If new mail has arrived during the *mail* session, *mail* displays a message:

You have new mail

If you **quit** while editing a mailbox file with the **−f** flag, *mail* rewrites the edit file. *Mail* then returns to the Shell, unless the rewrite of edit file fails, in which case the user can escape with the **exit** command.

reply (**r**) Takes a message list and sends mail to each message author just like the **mail** command. The default message must not be deleted.

replyall
Replies to all members on a distribution list, regardless of the setting of the **replyall** variable.

replysender
Reply to the sender of the message, regardless of the setting of the of the **replyall** variable.

Reply (upper-case **R**) replies to all members on a distribution list.

respond
A synonym for **reply**.

Respond
A synonym for **Reply**.

save (**s**) Takes a message list and a filename and appends each message in turn to the end of the file. The filename in quotes, followed by the line count and character count is echoed on the user's terminal.

set (**se**) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form:

option=value

or:

option

shell (**sh**) Invokes an interactive version of the shell.

size Takes a message list and prints out the size in characters of each message.

source (**so**) Accepts *mail* commands from a file.

subject
(**su**) works like the **from** command but works with subjects instead of message headers.

top Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to five.

type (**t**) A synonym for **print**.

unalias
Takes a list of names defined by **alias** commands and discards the remembered groups of users. The group names no longer have any significance.

undelete
(**u**) Takes a message list and marks each one as *not* being deleted.

unset Takes a list of option names and discards their remembered values; the inverse of **set**.

visual (**v**) Takes a message list and invokes the visual editor on each message. Note that the cautions given for the **edit** option above — about the header and final line of the message — apply here as well.

write (**w**) is similar to the **save** command, but **write** removes the 'From' line and the 'Status' line from the header of the messages, whereas **save** leaves those lines in the message.

xit (**x**) A synonym for **exit**.

z Scrolls forwards and backwards through groups of headers, 20 at a time (fewer lines of headers on low-speed terminals). If a '+' argument is given, the next 20 headers are displayed, and if a '–' argument is given, the previous 20 headers are displayed.

ESCAPE SEQUENCES

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name 'tilde escape' is somewhat of a misnomer since the actual escape character can be set by the option **escape**.

~!command
Execute the indicated shell command, then return to the message.

~c name ...
Add the given names to the list of carbon copy recipients.

~d Read the file *dead.letter* from your home directory into the message.

~e Invoke the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~f messages
Read the named messages into the message being sent. If no messages are specified, read the current message. Also see the **~m** command which does the same thing but shifts the message right by one tab stop.

~h Edit the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field by using the current terminal erase and kill characters.

~m messages
Read the named messages into the message being sent, shifted right one tab. If no messages are specified, read the current message.

~p Print out the message collected so far, prefaced by the message header fields.

~q Abort the message being sent, copying the message to *dead.letter* in your home directory if **save** is set.

~r filename
Read the named file into the message.

~s string
Make the named string become the current subject field.

~t name ...
Add the given names to the direct recipient list.

~v Invoke an alternate editor (defined by the VISUAL option) on the message collected so far. Usually, the alternate editor is a screen editor. After you quit the editor, you may resume appending text to the end of your message.

~w filename
Write the message onto the named file.

~|command
Pipe the message through the command as a filter. If the command gives no output or terminates abnormally, retain the original text of the message. The command *fmt*(1) is often used as *command* to rejustify the message.

~~string
: Insert the string of text in the message prefaced by a single ~. If you have changed the escape character, then you should double that character in order to send it.

MAIL OPTIONS

Options are controlled via the **set** and **unset** commands. Options may be either binary, in which case it is only significant to see whether they are set or not, or string, in which case the actual value is of interest. You can set and unset options during a *mail* session, or you can place the options in your *.mailrc* file in your home directory.

Binary Options. The binary options include the following:

append
Messages saved in *mbox* are appended (the default case) to the end rather than prepended. This is set in */usr/lib/Mail.rc* on version 7 systems.

ask *Mail* prompts you for the subject of each message you send. If you respond with simply a newline, no subject field is sent.

askcc *Mail* prompts for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

autoprint
The **delete** command behaves like **dp** — thus, after deleting a message, the next one is typed automatically.

ignore *Mail* ignores interrupt signals from your terminal and echoes them as @'s.

hold This option is used to hold messages in the system mailbox by default.

metoo Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting **metoo** includes the sender in the group.

nosave
Prevents *mail* from saving messages in the *dead.letter* file in your home directory on receipt of two interrupts (or after a **~q**). The default action is to save such messages in the *dead.letter* file.

quiet Suppress printing the version of the *mail* program when first invoked.

replyall
Alters the action of the **reply** and **Reply** commands: **reply** (lower-case **r**) normally replies to just the sender of the message. **Reply** (upper-case **R**) normally replies to everyone on the distribution list. Setting the **replyall** variable reverses this behavior so that **r** (lower-case **r**) replies to everyone on the distribution list, and **R** (upper-case **R**) replies to just the sender. This feature is here to retain compatibility with older versions of the *mail* system.

String Options. The following options have string values:

EDITOR
Pathname of the text editor to use in the **edit** command and ~e escape. If not defined, then a default editor is used. The default editor is */usr/ucb/ex*.

SHELL Pathname of the shell to use in the **!** command and the ~! escape. A default shell is used if this option is not defined. The default shell is */bin/csh*.

VISUAL
Pathname of the text editor to use in the **visual** command and ~v escape.

escape If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.

record If defined, gives the pathname of the file used to record all outgoing mail. If not defined, outgoing mail is not saved.

toplines
If defined, gives the number of lines of a message to be printed out with the **top** command; normally, the first five lines are printed.

crt If defined, gives the number of lines of a message to be printed before *mail* calls up the *more*(1) utility so that you can peruse the mail messages leisurely.

EXAMPLES

Send mail to Wendy on her system called ariel:

```
angel% mail wendy@ariel
Subject: New DBX Manuals
The new DBX manual pages are in angel:/usr/man/man1/dbx.1
^D
EOT
angel%
```

Note that *mail* asked for a subject; this is controlled by the **ask** option in the *.mailrc* file. There is an example of such a file later on in this section.

Now when Wendy is working on her system, she will see a mail notification:

```
work, work, work
You have new mail.
ariel% mail
Mail version 2.17  12/26/82.  Type ? for help
"/usr/spool/mail/wendy": 1 message 1 new
>N  1  cuthbert  Thu Nov  3 10:03:01  11/265  "New DBX Manual Pages"
&                                Mail signals its readiness for commands
                                 Carriage-return means read next message

From cuthbert Thu Nov  3 10:03:01 1983
Date: 3 Nov 83 10:02:56 PST (Thu)
From: cuthbert (Cuthbert Pouncetrifle)
Subject: New DBX Manual Pages
To: wendy
Status: R

The new DBX manual pages are in angel:/usr/man/man1/dbx.1

&                                Mail signals its readiness for commands
                                 Carriage-return means read next message
At EOF                           Means there are no more messages
& d                              Delete this message
& q                              Quit from the mail program
ariel%
```

Here is a *.mailrc* file in which you can set options to control *mail*'s behavior:

```
set crt=30 ask
ignore via apparently-to date from status received message-id
```

The first line sets the **crt** option to 30 lines — if there are messages longer than this, *mail* calls up *more* to page through the message in 30-line chunks. The **ask** option makes *mail* ask for a subject heading every time you send mail to someone. Finally, the second line says to ignore any line starting with any of the character strings from the list — this shortens the amount of irrelevant junk that appears at the top of mail messages.

Here is a line from a *.login* file (oriented to the C-Shell):

```
set mail=(10 /usr/spool/mail/$USER)
```

This **set** command instructs the C-Shell to look in */usr/spool/mail/$USER* for mail every 10 seconds and to notify you if there is mail waiting there. In the absence of the time specification, the Shell looks every five minutes.

FILES

/usr/spool/mail/*	post office
~/mbox	your old mail
~/.mailrc	file giving initial mail commands
/tmp/R#	temporary for editor escape
/usr/lib/Mail.help*	help files
/usr/lib/Mail.rc	system initialization file
/bin/mail	to do actual mailing
/usr/lib/sendmail	postman

SEE ALSO

binmail(1), fmt(1), newaliases(8), aliases(5), sendmail(8), mailaddr(7), biff(1)
Mail User's Guide in the *Beginner's Guide to the Sun Workstation*, which gives an excellent introduction.
For *sendmail* installation instructions, see the Sun *System Manager's Manual*.

NAME

make – maintain program groups

SYNOPSIS

make [**–f** *makefile*] [**–i**] [**–k**] [**–n**] [**–t**] [**–r**] [**–s**] [**–d**] [**–p**] [**–S**] [**–q**] file ...

DESCRIPTION

Make executes commands in *makefile* to update one or more target *files*. *File* is typically a program. If no **–f** option is present, 'makefile' and 'Makefile' are tried in order. 'Makefile' with a capital M is the preferred choice since it appears towards the front of any list of files and thus tends to stand out. If *makefile* is '–', *make* reads the standard input. More than one **–f** option may appear.

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

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target. If a name appears on the left of more than one 'colon' line, then it depends on all of the names on the right of the colon on those lines, but only one command sequence may be specified for it. If a name appears on a line with a double colon **::** then the command sequence following that line is performed only if the name is out of date with respect to the names to the right of the double colon, and is not affected by other double colon lines on which that name may appear.

Two special forms of a name are recognized. A name like *library*(*file*) means the file named *file* stored in the archive named *library*. A name like *library*((*entry*)) means the file stored in archive *library* containing the entry point *entry*.

Sharp and newline surround comments, except on lines containing shell commands.

The following makefile says that *program* depends on the two files *a.o* and *b.o* and that they in turn depend on *.c* files and a common file called *incl*:

```
program: a.o b.o
        cc a.o b.o –lm –o program
a.o: incl a.c
        cc –c a.c
b.o: incl b.c
        cc –c b.c
```

Makefile entries of the form

```
string1 = string2
```

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

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

```
program: a.o b.o
        cc a.o b.o –lm –o program
a.o b.o: incl
```

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

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

The rule to create a file with suffix *s2* that depends on a similarly named file with suffix *s1* is specified as an entry for the 'target' *s1s2*. Before issuing any command, *make* sets certain special macros:

$@ is set to the name of the file to be 'made'.

$? is set to the names of files that are younger than the target.

$< is set to the name of the related file that caused the action, only if the command was generated by an implicit rule.

$* is set to the prefix shared by the current and dependent filenames, only if the command was generated by an implicit rule.

For example, a rule for making optimized *.o* files from *.c* files is

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

Certain macros are used by the default inference rules to communicate optional arguments to any resulting compilations. In particular, 'CFLAGS' is used for *cc*(1) options, 'FFLAGS' for *f77*(1) options, 'PFLAGS' for *pc*(1) options, and 'LFLAGS' and 'YFLAGS' for *lex* and *yacc*(1) options. In addition, the macro 'MFLAGS' is filled in with the initial command line options supplied to *make*. This simplifies maintaining a hierarchy of makefiles as one may then invoke *make* on makefiles in subdirectories and pass along useful options such as **−k**.

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

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

Interrupt and quit cause the target to be deleted unless the target is a directory or depends on the special name .PRECIOUS.

OPTIONS

−f *myfile*
myfile is the name of the file to use for *make* commands instead of the default 'makefile' or 'Makefile'.

−i Equivalent to the special entry .IGNORE:.

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

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

−t Touch, that is, update the modified date of targets, without executing any commands.

−r Equivalent to an initial special entry .SUFFIXES: with no list.

−s Equivalent to the special entry .SILENT:.

−d debug; print internal state after each command

−p print internal state after reading Makefile

−q determines if the target is up to date — returns a zero exit status if it is, and a non-zero exit status otherwise.

−S undoes the effect of the **−k** option.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1), f77(1), pc(1)
Make — A Program for Maintaining Computer Programs, in *Programming Tools for the Sun System*.

BUGS

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

Make should recognize names of libraries in the dependency entries, for example:

```
grab:   -lm  grab.o
        cc  -o  grab  grab.o  -lm
```

NAME

man – print out manual pages; find manual information by keywords

SYNOPSIS

man [–] [**–t**] [**–P** *path*] [section] title ...
man –k *keyword* ...
man –f *file* ...

DESCRIPTION

Man displays information from the UNIX System manual pages. *Man* can be asked for one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. *Man* can also provide on-line access to the sections of the printed manual.

Man's standard behavior when neither the **–k** nor **–f** option is specified (see OPTIONS below) is to format a specified set of manual pages. In the absence of any specific options to the contrary, *man* looks for already formatted manuals in the */usr/man/cat*? directories and uses the information cached there. Otherwise, *man* must format the pages on the fly — in this case you get a message asking you to wait patiently.

If *section* is omitted, *man* searches all sections of the manual, giving preference to commands over subroutines in system libraries, and prints the first section it finds, if any.

If a *section* is specified, *man* looks in that section of the manual for the given *titles*. *Section* is an arabic section number ('3' for instance); the number may be followed by a single letter classifier ('1g', for instance, indicating a graphics program in Section 1). Finally, *section* can be one of the keywords 'public', 'local', 'old', or 'new'.

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

OPTIONS

–P *path*
Use *path* as the starting pathname for the manual pages. Manual page sections must reside in *path*/man? instead of in */usr/man*/man?

–k *keywords*
Display a one line synopsis of each manual section whose listing in the table of contents contains any of the *keywords*.

–f *filenames*
Attempt to locate manual sections related to those files, and display the table of contents lines for those sections.

–t Use *troff* to format the specified section, assuming you have a suitable raster output device which can actually handle *troff*'s output.

FILES

/usr/man/man?/*	manual pages in *nroff*/*troff* source
/usr/man/cat?/*	formatted manual pages

SEE ALSO

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

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), talk(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 directories. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

The current *umask*(2) setting determines the mode in which directories are created. Modes may be modified after creation by using *chmod*(1).

Mkdir requires write permission in the parent directory.

SEE ALSO

chmod(1), rmdir(1), rm(1), umask(2), mkdir(2), rmdir(2)

NAME

mkstr – create an error message file by massaging C source

SYNOPSIS

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr creates files of error messages. You can use *mkstr* to make programs with large numbers of error diagnostics much smaller, and to reduce system overhead in running the program — as the error messages do not have to be constantly swapped in and out.

Mkstr processes each of the specified *files,* placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical example of using *mkstr* would be:

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx.*

To process the error messages in the source to the message file, *mkstr* keys on the string ‘error("’ in the input stream. Each time it occurs, the C string starting at the ‘"’ is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, that is:

```
char    efilname[] = "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

OPTIONS

– Place error messages at the end of the specified message file for recompiling part of a large *mkstr*'d program.

SEE ALSO

lseek(2), xstr(1)

NAME

more, page — browse through a text file

SYNOPSIS

more [**−cdflsu**] [*−lines*] [*+linenumber*] [*+/pattern*] [name ...]

page [**−cdflsu**] [*−lines*] [*+linenumber*] [*+/pattern*] [name ...]

DESCRIPTION

More is a filter which displays the contents of a text file one screenful at a time on a video terminal. It normally pauses after each screenful, and prints '--More--' at the bottom of the screen. *More* displays another line if you type a carriage-return; *more* displays another screenful if you type a space.

If you use the *page* command instead of the *more* command, the screen is cleared before each screenful is displayed (but only if a full screenful is being displayed), and $k - 1$ rather than $k - 2$ lines are displayed in each screenful, where k is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the **−c** mode of operation, the *csh* command *"setenv MORE -c"* or the *sh* command sequence *"MORE='-c' ; export MORE"* would cause all invocations of *more* , including invocations by programs such as *man* to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.login* or *.profile* file.

If *more* is reading from a file, rather than a pipe, a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
display *i* more lines, (or another screenful if no argument is given)

^D display 11 more lines (a "scroll"). If *i* is given, the scroll size is set to *i*.

d same as ^D (control-D)

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

*i*s skip *i* lines and print a screenful of lines

*i*f skip *i* screenfuls and print a screenful of lines

q or Q Exit from *more.*

= Display the current line number.

v Start up the editor *vi* at the current line.

h Help command; give a description of all the *more* commands.

i/expr search for the *i*-th occurrence of the regular expression *expr.* If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found, or the end of the pipe, whichever comes first. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
: invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q
: exit from *more* (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately; it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may type the line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can type the quit key (normally control—\). *More* stops sending output, and displays the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

More sets the terminal to *noecho* mode so that the output can be continuous. Thus what you type does not show on your terminal, except for the / and ! commands.

If the standard output is not a terminal, *more* acts just like *cat,* except that a header is printed before each file in a series.

OPTIONS

—lines Set the size of the window to *lines* lines long instead of the default.

—c Display each page by redrawing the screen instead of scrolling. This makes it easier to read text while *more* is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

—d Display the message 'Hit space to continue, Rubout to abort' at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where users are unsophisticated.

—f Count logical rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul,* since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen postions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

—l Do not treat ^L (form feed) specially. If *—l* is not used, *more* pauses after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

−s Squeeze multiple blank lines from the output, and replace them with single blank lines. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

−u Normally, *more* handles underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *−u* option suppresses this processing.

+*linenumber*
Start up at *linenumber*.

+/*pattern*
Start up two lines before the line containing the regular expression *pattern*.

EXAMPLES

A sample usage of *more* in previewing *nroff* output would be

```
nroff −ms +2 doc.n ¦ more -s
```

FILES

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

SEE ALSO

csh(1), man(1), script(1), sh(1), environ(5), termcap(5)

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**–f** *tapename*] *command* [*count*]

DESCRIPTION

mt sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable TAPE is used; if TAPE does not exist, *mt* uses the device */dev/rmt12*. Note that *tapename* must reference a raw (not block) tape device. By default *mt* performs the requested operation once. Operations may be performed multiple times by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

eof, weof
Write *count* end-of-file marks at the current position on the tape.

fsf Forward space *count* files.

fsr Forward space *count* records.

bsf Back space *count* files.

bsr Back space *count* records.

For the following commands, *count* is ignored:

rewind
Rewind the tape.

offline, rewoffl
Rewind the tape and place the tape unit off-line.

status Print status information about the tape unit.

retension
Retension the tape.

erase Erase the entire tape.

mt returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

FILES

/dev/rmt*	Raw magnetic tape interface
/dev/rar*	Raw Archive cartridge tape interface
/dev/rst*	Raw SCSI tape interface

SEE ALSO

mtio(4), dd(1), ioctl(2), environ(5)

BUGS

Not all devices support all commands. For example, *ar*(4s) and *st*(4s) currently do not support the **fsr, bsf,** or **bsr** commands; but they are the only only ones that currently support the **retension** and **rewind** commands.

NAME

mv − move or rename files

SYNOPSIS

mv [**−i**] [**−f**] [**−**] file1 file2

mv [**−i**] [**−f**] [**−**] directory1 directory2

mv [**−i**] [**−f**] [**−**] file ... directory ... directory

DESCRIPTION

Mv moves files and directories around in the file system. A side effect of *mv* is to rename a file or directory. The three major forms of *mv* are shown in the synopsis above.

The first form of *mv* moves (changes the name of) *file1* to *file2*. If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place, otherwise *mv* exits.

The second form of *mv* moves (changes the name of) *directory1* to *directory2*, *ONLY* if *directory2* does not already exist — if it does, the third form applies.

The third form of *mv* moves one or more *files* and *directories*, with their original names, to the last *directory* in the list.

Mv refuses to move a file or directory onto itself.

OPTIONS

−i interactive mode: *mv* displays the name of the file or directory followed by a question mark whenever a move would replace an existing file or directory. If you type a line starting with 'y', *mv* moves the specified file or directory, otherwise *mv* does nothing with that file or directory.

−f force: override any mode restrictions and the **−i** switch. The **−f** option also suppresses any warning messages about modes which would potentially restrict overwriting.

− Interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

SEE ALSO

cp(1), ln(1)

BUGS

If *file1* and *file2* are on different file systems, then *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Mv will not move a directory from one file system to another.

NAME

nice, nohup – run a command at low priority (*sh* only)

SYNOPSIS

nice [–*number*] command [arguments]

nohup command [arguments]

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. '––10'.

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. *Nohup* should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal. The syntax of nice is also different.

FILES

nohup.out	standard output and standard error file under *nohup*

SEE ALSO

csh(1), nice(3C), renice(8)

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

Nice and *nohup* are particular to *sh*(1). If you use *csh*(1), then commands executed with "&" are automatically immune to hangup signals while in the background. There is a builtin command *nohup* which provides immunity from terminate, but it does not redirect output to *nohup.out*.

Nice is built into *csh*(1) with a slightly different syntax than described here. The form "nice +10" nices to positive nice, and "nice –10" can be used by the super-user to give a process more of the processor.

NAME

nm – print name list

SYNOPSIS

nm [**–gnoprua**] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in *a.out* are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters:

U	(undefined),
A	(absolute),
T	(text segment symbol),
D	(data segment symbol),
B	(bss segment symbol),
C	(common symbol),
f	file name,
–	debug symbol table entries (see **–a** below).

If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

OPTIONS

–g Print only global (external) symbols.

–n Sort numerically rather than alphabetically.

–o Prepend file or archive element name to each output line rather than only once.

–p Don't sort; print in symbol-table order.

–r Sort in reverse order.

–u Print only undefined symbols.

–a Print all symbols.

EXAMPLE

```
nm
```

prints the symbol list of *a.out,* the default output file for the C compiler.

SEE ALSO

ar(1), ar(5), a.out(5)

NAME

nroff − text formatting and typesetting

SYNOPSIS

nroff [**−o***pagelist*] [**−n***N*] [**−s***N*] [**−m***name*] [**−r***aN*] [**−i**] [**−q**] [**−T***name*]
[**−e**] [**−h**] [file] ...

DESCRIPTION

Nroff formats text in the named *files* for typewriter-like devices. See also *troff*(1). The full capabilities of *nroff* and *troff* are described in *Formatting Documents with Nroff and Troff.*

If no *file* argument is present, *nroff* reads the standard input. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear *before* the files.

−o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end.

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.

−m*name*
Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

−T*name*
Prepare output for a device of the specified *name*. Known *names* are:

37	Teletype Corporation Model 37 terminal — this is the default.
tn300	GE TermiNet 300 (or any terminal without half-line capability),
300S	DASI-300S
300	DASI-300
300X	DASI-300X
450	DASI-450 (Diablo Hyterm).
450-12	DASI-450 (Diablo Hyterm) — 12-pitch.
450-12-8	DASI-450 (Diablo Hyterm) — 12-pitch and 8 lines-per-inch.
450X	DASI-450X (Diablo Hyterm).
lpr	Line printer.
x1700	Xerox 1700 daisywheel printer.

−e Produce equally-spaced words in adjusted lines, using full terminal resolution.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

EXAMPLE

```
gaia% nroff −s4 −me users.guide
```

Formats *users.guide* using the **−me** macro package, and stopping every 4 pages.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files

/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

Formatting Documents with Nroff and Troff in *Editing and Text Processing on the Sun Workstation.*
troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1), checknr(1)

NAME

od — octal, decimal, hex, ascii dump

SYNOPSIS

od [−format] [file] [[**+**]offset[**.**][**b**] [label]]

DESCRIPTION

Od displays *file*, or it's standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **−o** (octal) is the default. Dumping continues until end-of-file.

The meanings of the format argument characters are:

a Interpret bytes as characters and display them with their ASCII names. If the **p** character is given also, bytes with even parity are underlined. If the **P** character is given, bytes with odd parity are underlined. Otherwise the parity bit is ignored.

b Interpret bytes as unsigned octal.

c Interpret bytes as ASCII characters. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.

d Interpret (short) words as unsigned decimal.

f Interpret long words as floating point.

h Interpret (short) words as unsigned hexadecimal.

i Interpret (short) words as signed decimal.

l Interpret long words as signed decimal.

o Interpret (short) words as unsigned octal.

s[n] Look for strings of ASCII graphic characters, terminated with a null byte. *N* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.

v Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an "*" in column 1.

w[n] Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

x Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; If "**.**" is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with "**x**" or "**0x**", it is interpreted in hexadecimal. If "**b**" ("**B**") is appended, the offset is interpreted as a block count, where a block is 512 (1024) bytes. If the *file* argument is omitted, an *offset* argument must be preceded by "**+**".

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

Label will be interpreted as a pseudo-address for the first byte displayed. It will be shown in "()" following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

SEE ALSO

adb(1)

BUGS

A file name argument can't start with "+". A hexadecimal offset can't be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

NAME

pagesize – print system page size

SYNOPSIS

pagesize

DESCRIPTION

Pagesize prints the size of a page of memory in bytes, as returned by *getpagesize*(2). This program is useful in constructing portable shell scripts.

SEE ALSO

getpagesize(2)

NAME

passwd – change login password

SYNOPSIS

passwd [**–f** *filename*] [*user-ID*]

DESCRIPTION

This command changes (or installs) a password associated with the *user-ID* (your own by default).

Passwd prompts for the old password and then for the new one. You must supply both, and the new password must be typed twice to forestall mistakes.

New passwords must be at least four characters long, if they use a sufficiently rich alphabet, or at least six characters long if in monocase.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

OPTIONS

–f Treat *file* as the password file.

FILES

/etc/passwd
/etc/yp/passwd

SEE ALSO

login(1), passwd(5), crypt(3), yppasswd(1)
Robert Morris and Ken Thompson, *UNIX Password Security*

BUGS

Passwd will not change your password if your system uses the *yp* Yellow Pages name server. Refer to *yppasswd*(1) for more information.

NAME

pc − Pascal compiler

SYNOPSIS

pc [**−c**] [**−g**] [**−o** *output*]] **−O**] [**−b**] [**−C**] [**−fsky**] [**−H**] [**−i** *name* ...] [**−l**]
[**−lpfc**] [**−L**]] **−P**] [**−s**] [**−S**]] **−z**] *filename.p* ...

DESCRIPTION

Pc is the Sun Pascal compiler. If given an argument file ending with *.p, pc* compiles the file and leaves the result in an executable file called *a.out* by default.

A program may be separated into more than one *.p* file. *Pc* will compile a number of *.p* files into object files (with the extension *.o* in place of *.p*). Object files may then be loaded into an executable *a.out* file. Exactly one object file must supply a **program** statement to successfully create an executable *a.out* file. The rest of the files must consist only of declarations which logically nest within the program. References to objects shared between separately compiled files are allowed if the objects are declared in **included** header files, whose names must end with *.h*. Header files may only be included at the outermost level, and thus declare only globally available objects. To allow external **functions** and **procedures** to be declared, an **external** directive has been added, whose use is similar to the **forward** directive but restricted to appear only in *.h* files. **Function** and **procedure** bodies may not appear in *.h* files. A binding phase of the compiler checks that declarations are used consistently, to enforce the type checking rules of Pascal.

Object files created by other language processors may be loaded together with object files created by *pc*. The **functions** and **procedures** they define must have been declared in *.h* files included by all the *.p* files which call those routines.

Pascal's calling conventions are the same as in C, with **var** parameters passed by address and other parameters passed by value.

Both *pc* and *pi*(1) support ISO Level 1 Standard Pascal, including conformant array parameters. Deviations from the ISO Standard are noted under **BUGS** below.

See the *Pascal User's Manual* for details.

OPTIONS

See *ld*(1) for load-time options.

−c Suppress loading and produce *.o* file(s) from source file(s).

−g Produce additional symbol table information for the symbolic debugger *dbx*(1).

−o *name*
Name the final output file *name* instead of *a.out*.

−O Optimize the object code.

−b Buffer the file *output* in units of disk blocks, rather than lines.

−C Compile code to perform subscript and subrange checks, verify **assert** statements, and initialize all variables to zero as in *pi*. Note that pointers are not checked. This option differs significantly from the **-C** option of the *cc* compiler.

−fsky Generate code which assumes the presence of a SKY floating-point processor board. Programs compiled with this option can only be run in systems that have a SKY board installed. Programs compiled without the **−fsky** option will use the SKY board from library routines, but won't run as fast as they would if the **−fsky** option were used. If any part of a program is compiled using the **−fsky** option, you must also use this option when linking with the **pc** command, since different startup routines are used to initialize the SKY board.

−H Compile code to perform range checks on pointers into the heap.

−i *name*

Produce a listing for the specified procedures, functions and **include** files.

−l Make a program listing during translation.

−lpfc Load common startup code for programs containing mixed Pascal and FORTRAN object files. Such programs should also be loaded with the FORTRAN libraries (see **files** below).

−L Map upper case letters in keywords and identifiers to lower case.

−P Use partial evaluation semantics for the boolean operators **and** and **or**. For these operators only, left-to-right evaluation is guaranteed, and the second operand is evaluated only if necessary to determine the result.

−s Accept standard Pascal only; nonstandard constructs cause warning diagnostics.

−S Compile the named program, and leave the assembly language output on the corresponding file suffixed '.s'. No '.o' is created.

−z Allow execution profiling with *pxp* by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

Other arguments are taken to be loader option arguments or libraries of *pc* compatible routines. Certain flags can also be controlled in comments within the program, as described in the *Pascal User's Manual* in the Sun *Pascal* Manual.

FILES

file.p	Pascal source files
/lib/cpp	macro preprocessor
/usr/lib/pc0	compiler
/lib/f1	code generator
/usr/lib/pc2	inline expander of library calls
/lib/c2	peephole optimizer
/usr/lib/pc3	separate compilation consistency checker
/usr/lib/pc3.2strings	text of the error messages
/usr/lib/how_pc	basic usage explanation
/usr/lib/libpc.a	intrinsic functions and I/O library
/usr/lib/libpfc.a	startup code for combined Pascal and FORTRAN programs
/usr/lib/libF77.a	FORTRAN intrinsics library
/usr/lib/libI77.a	FORTRAN I/O library
/usr/lib/libU77.a	FORTRAN<=>Unix interface library
/usr/lib/libm.a	math library
/lib/libc.a	standard library, see *intro*(3)

SEE ALSO

The *Pascal User's Manual* in the Sun *Pascal* Manual.
pi(1), pxp(1), pxref(1)

DIAGNOSTICS

For a basic explanation do

```
    tutorial% pc
```

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.

Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow-up messages begin with an ellipsis '...'.

The first character of each error message indicates its class:

E	Fatal error; no code will be generated.
e	Nonfatal error.
w	Warning – a potential problem.
s	Nonstandard Pascal construct warning.

If a severe error occurs which inhibits further processing, the translator will give a diagnostic and then 'QUIT'.

Names whose definitions conflict with library definitions draw a warning. The library definition will be replaced by the one supplied in the Pascal program. Note that this can have unpleasant sideeffects.

BUGS

The keyword **packed** is recognized but has no effect. The ISO standard requires packed and unpacked structures to be distinguished for portability reasons.

Binary set operators are required to have operands with identical types; the ISO standard allows different types, as long as the underlying base types are compatible.

The **−z** flag doesn't work for separately compiled files.

Because the **−s** option is usurped by the compiler, it is not possible to pass the strip option to the loader. Thus programs which are to be stripped, must be run through *strip*(1) after they are compiled.

NAME

perfmeter – meter display of system performance values

SYNOPSIS

perfmeter [**–s** *sampletime*] [**–h** *hourhandintv*] [**–m** *minutehandintv*] [**–v** *value*] [*hostname*]

DESCRIPTION

Perfmeter starts a tool whose iconic form is a meter displaying a system performance value, and whose open form is a strip chart of that value. The default is for the meter to be updated with a sampletime of 2 seconds, for the hour hand to represent an average over an interval of 20 seconds, for the minute hand to represent an average over 2 seconds, and for the value being displayed to be percent of cpu being utilized. These defaults can be modified with command line arguments, or dynamically after the tool has been started. If there is no *hostname* argument, then the data displayed will be for the local machine, otherwise for the machine named by *hostname*. In either case, the *rstatd(8)* daemon must be running on the machine for which statistics are being reported.

OPTIONS

–s *sampletime*
Sets the sample time to *sampletime* seconds.

–h *hourhandintv*
Sets the hour hand interval to *hourhandintv* seconds.

–m *minutehandintv*
Sets the minute hand interval to *minutehandintv seconds.*

–v *value*
Set the performance value to be monitored to one of:

cpu	percent of cpu being utilized
pkts	ethernet packets per second
page	paging activity in pages per second
swap	jobs swapped per second
disk	disk traffic in transfers per second
intr	number of device interrupts per second
cntxt	number of context switches per second

COMMANDS

The value being displayed can be changed by clicking the tool with the rightmost mouse button, and then selecting the appropriate menu item. The other meter parameters can be modified by moving the mouse cursor into the tool (either open or closed), and typing:

m	decreases *minutehandintv* by one
M	increases *minutehandintv* by one
h	decreases *hourhandintv* by one
H	increases *hourhandintv* by one
1-9	Sets *sampletime* to a range from 1 to 9 seconds.

FILES

/etc/servers	starts statistics server

SEE ALSO

perfmon(1), netstat(8), vmstat(8), suntools(1), rstatd(8)

NAME

perfmon – graphical display of general system statistics

SYNOPSIS

perfmon [statistic **...**]

DESCRIPTION

Perfmon provides a graphical display of the system-wide performance statistics and updates them approximately once a second. It should be executed from a graphics tool inside the SunWindows system. The time interval between updates can be adjusted by typing one of the following characters while the mouse is in the graphics subwindow:

s increases the interval by 0.05 seconds (small *s* means get *s*lower by a little).

S increases the interval by one second (capital *S* means get *S*lower by a lot).

f decreases the interval by 0.05 seconds (small *f* means get *f*aster by a little).

F decreases the interval by one second (capital *F* means get *F*aster by a lot).

R resets the interval to the standard 1.05 seconds.

In addition, typing:

H, **h** or **?**
lists the characters that *perfmon* is listening for.

Q or **q** causes *perfmon* to cease executing.

If no statistic argument is given, *perfmon* displays all statistics. A tick is placed on the lines separating the graphs once every fifteen seconds (due to scheduling vagaries, these ticks may not be evenly spaced).

Statistics

user is the percentage of total CPU time spent in normal and low priority user processes.

system
is the percentage of total CPU time attributed to system calls and overhead.

idle is the percentage of total CPU time spent idle.

free is the amount of available real memory (in Kbytes).

disk is the total number of disk transfers performed.

interrupts
is the total number of interrupts serviced.

input is the total number of input packets received.

output
is the total number of output packets transmitted.

collision
is the total number of collisions between packets observed on the network.

SEE ALSO

netstat(8), perfmeter(1), suntools(1), vmstat(8)

BUGS

Perfmon should use *rstatd*.

NAME

pi — Pascal interpreter code translator

SYNOPSIS

pi [**−b**] [**−l**] [**−L**] [**−n**] [**−o** *name*] [**−p**] [**−s**] [**−t**] [**−u**] [**−w**] [**−z**]
[**−i** name **...**] name.p

DESCRIPTION

Pi translates the program in the file *name.p* leaving interpreter code in the file *obj* in the current directory. The interpreter code can be executed using *px*. *Pix* performs the functions of *pi* and *px* for 'load and go' Pascal.

Both *pi* and *pc*(1) support ISO Level 1 Standard Pascal, including conformant array parameters. Deviations from the ISO Standard are noted under **BUGS** below.

OPTIONS

The following flags are interpreted by *pi;* the associated options can also be controlled in comments within the program; see the *Pascal User's Manual* in the *Sun Fortran and Pascal* Manual for details.

−b Buffer the file *output* in units of disk blocks, rather than lines.

−i *name*
Enable the listing for any specified procedures, functions, and **include** files.

−l Make a program listing during translation.

−L Map all identifiers and keywords to lower case.

−n Begin each listed **include** file on a new page with a banner line.

−o *name*
Name the final output file *name* instead of *a.out.*

−p Suppress the post-mortem control flow backtrace if an error occurs; suppress statement limit counting.

−s Accept standard Pascal only; non-standard constructs cause warning diagnostics.

−t Suppress runtime tests of subrange variables and treat **assert** statements as comments.

−u Card image mode; only the first 72 characters of input lines are used.

−w Suppress warning diagnostics.

−z Allow execution profiling with *pxp* by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

FILES

file.p	input file
file.i	**include** file(s)
/usr/lib/pi3.*strings	text of the error messages
/usr/lib/how_pi*	basic usage explanation
obj	interpreter code output

SEE ALSO

Sun Fortran and Pascal Manual
pix(1), px(1), pxp(1), pxref(1)

DIAGNOSTICS

For a basic explanation do

```
tutorial% pi
```

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.

Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow-up messages begin with an ellipsis '...'.

The first character of each error message indicates its class:

E	Fatal error; no code will be generated.
e	Non-fatal error.
w	Warning — a potential problem.
s	Non-standard Pascal construct warning.

If a severe error occurs which inhibits further processing, the translator will give a diagnostic and then 'QUIT'.

BUGS

The keyword **packed** is recognized but has no effect. The ISO standard requires packed and unpacked structures to be distinguished for portability reasons.

Binary set operators are required to have operands with identical types; the ISO standard allows different types, as long as the underlying base types are compatible.

For clarity, semantic errors should be flagged at an appropriate place in the source text, and multiple instances of the 'same' semantic error should be summarized at the end of a **procedure** or **function** rather than evoking many diagnostics.

When **include** files are present, diagnostics relating to the last procedure in one file may appear after the beginning of the listing of the next.

NAME

pix – Pascal translator and interpreter

SYNOPSIS

pix [options] [**−i** name ...] name.p [argument ...]

DESCRIPTION

Pix is a 'load and go' version of Pascal which combines the functions of the translator *pi* and the interpreter *px*. *Pix* uses *pi* to translate the program in the file *name.p* and, if there were no fatal errors during translation, calls *px* to execute the resulting interpretive code with the specified arguments. A temporary file is used for the object code; the file *obj* is neither created nor destroyed.

Options are as described under pi(1).

FILES

/usr/ucb/pi	Pascal translator
/usr/ucb/px	Pascal interpreter
/tmp/pix*	temporary
/usr/lib/how_pix	basic explanation

SEE ALSO

The *Pascal User's Manual* in the *Pascal for the Sun Workstation* Manual.
pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

```
tutorial% pix
```

NAME

plot − graphics filters

SYNOPSIS

plot [**−T**terminal [raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(5)) from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter TERM (see *environ*(5)) is used. Known *terminals* are:

4014 Tektronix 4014 storage scope.

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

ver Versatec D1200A printer-plotter. This version of *plot* places a scan-converted image in */usr/tmp/raster* and sends the result directly to the plotter device rather than to the standard output. The optional argument sends a previously scan-converted file called *raster* to the plotter.

FILES

/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/tmp/raster

SEE ALSO

plot(3X), plot(5)

BUGS

There is no lockout protection for */usr/tmp/raster*.

NAME

pmerge — pascal file merger

SYNOPSIS

pmerge name.p ...

DESCRIPTION

Pmerge assembles the named Pascal files into a single standard Pascal program. The resulting program is listed on the standard output. It is intended to be used to merge a collection of separately compiled modules so that they can be run through **pi**, or exported to other sites.

FILES

/usr/tmp/MG*	default temporary files

SEE ALSO

pc(**1**), pi(1),
Auxiliary documentation *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.

BUGS

Very minimal error checking is done, so incorrect programs will produce unpredictable results. Block comments should be placed after the keyword to which they refer or they are likely to end up in bizarre places.

NAME

postnews – submit news articles

SYNOPSIS

postnews [*article*]

DESCRIPTION

Postnews calls up the *inews*(1) utility to submit news articles to USENET. *Postnews* prompts you for the title of the article, for the newsgroup, and for the distribution. The title of the article should be a phrase suggesting the subject, so that persons reading the news can tell if they are interested in the article.

If you omit the name of the newsgroup (by typing a carriage-return when asked for the newsgroup), *postnews* posts the article to *general.*

general is read by everyone on the local machine. Other possible newsgroups include, but are not limited to, **btl.general,** read by all users at all Bell Labs sites on USENET, **net.general,** read by all users at all sites on USENET, and **net.news,** read by users interested in the network news on all sites. There is often a local set of newsgroups, such as **ucb.all,** that circulate within a local set of machines. In this case, *ucb* newsgroups circulate among machines at the University of California at Berkeley.

The distribution can be any valid list of newsgroup names, and defaults to the same as the newsgroup. If they are the same, the distribution is omitted from the headers put into the editor buffer. A distribution header is included in the headers of the article if given, affecting where the article is distributed to.

After entering the title, newsgroup, and distribution, *postnews* calls up an editor program where you can compose the article. *Postnews* uses *vi*(1) as the editor unless you have specified another editor via the $EDITOR environment variable, in which case *postnews* uses the editor specified there.

An initial set of headers containing the subject and newsgroups will be placed in the editor, followed by a blank line. The article should be appended to the buffer, after the blank line. These headers can be changed, or additional headers added, while in the editor, if desired.

Optionally, the article is read from the file specified by *article.*

For more sophisticated uses, such as posting news from a program, see *inews*(1).

SEE ALSO

mail(1), checknews(1), inews(1), readnews(1).

NAME

pr − print file(s), possibly in multiple columns

SYNOPSIS

pr [−*n*] [**+***n*] [**−h** *string*] [**−w***n*] [**−f**] [**−l***n*] [**−t**] [**−s***n*] [**−m**] [file] ...

DESCRIPTION

Pr prepares one or more *files*'s for printing. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. *Pr* prints its standard input if there are no *file* arguments.

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

OPTIONS

Options apply to all following *file*'s but may be reset between *file*'s:

−*n* Produce *n*-column output. This option overrides the **−t** option (see below).

+*n* Begin printing with page *n*.

−h *string*
Use *string* as a header for the page instead of the default header.

−w*n* For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.

−f Use formfeeds instead of newlines to separate pages. A formfeed is assumed to use up two blank lines at the top of a page. Thus this option does not affect the effective page length.

−l*n* Take the length of the page to be *n* lines instead of the default 66.

−t Do not print the 5-line header or the 5-line trailer normally supplied for each page. Formfeed characters are not generated when this option is used, even if the **−f** option was used. The **−t** option is intended for applications where the results should be directed to a file for further processing.

−s*c* Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.

−m Print all *file*'s simultaneously, each in one column,

EXAMPLES

Print a file called *dreadnaught* on the printer — this is the simplest use of *pr*:

```
krypton% pr  dreadnaught | lpr
krypton%
```

Produce three laminations of a file called *ridings* side by side in the output, with no headers or trailers, the results to appear in the file called *Yorkshire*:

```
krypton% pr  −m −t ridings ridings ridings > Yorkshire
krypton%
```

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(1), lpr(1)

DIAGNOSTICS

There are no diagnostics when *pr* is printing on a terminal.

BUGS

The options described above interact with each other in strange and as yet to be defined ways.

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

prmail – print out waiting mail

SYNOPSIS

prmail [user ...]

DESCRIPTION

Prmail prints the mail which waits for you, or the specified *users*. The mail is not disturbed.

FILES

/usr/spool/mail/*	waiting mail files

SEE ALSO

biff(1), mail(1), from(1), binmail(1)

NAME

prof – display profile data

SYNOPSIS

prof [**–a**] [**–l**] [**–n**] [**–s**] [**–v** [*–low* [*–high*]]] [**–z**] [*object-file* [*profile-file* ...]]

DESCRIPTION

Prof interprets the file produced by the *monitor*(3) subroutine. In the default case, the symbol table in the named object file (*object-file* by default) is read and correlated with the profile file (*profile-file* by default). For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call. If more than one profile file is specified, the output represents the sum of the profiles.

To tally the number of calls to a routine, the program must be compiled with the **–p** option of *cc*, *f77* or *pc*. This option also means that the profile file is produced automatically.

Beware of quantization errors.

OPTIONS

–a Report all symbols rather than just external symbols.

–l Sort the output by symbol value.

–n sort the output by number of calls.

–s Produce a summary profile file in *mon.sum*. This is really only useful when more than one profile file is specified.

–v [*–low* [*–high*]]
Suppress all printing and produce a graphic version of the profile on the standard output for display by the *plot*(1G) filters. When plotting, the numbers *low* and *high*, (by default 0 and 100), select a percentage of the profile to be plotted, with accordingly higher resolution.

–z Print routines which have zero usage (as indicated by call counts and accumulated time).

FILES

mon.out	for profile
a.out	for namelist
mon.sum	for summary profile

SEE ALSO

monitor(3), profil(2), cc(1), plot(1G), gprof(1)

BUGS

Prof is confused by *f77* which puts the entry points at the bottom of subroutines and functions.

NAME

prs – print an SCCS file

SYNOPSIS

/usr/sccs/prs [**–d**[*dataspec*]] [**–r**[*SID*]] [**–e**] [**–l**] [**–a**] file ...

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of – is given, the standard input is read, in which case each line is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

OPTIONS

Options apply independently to each named file.

–d[*dataspec*]
Specifies the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

–r[*SID*]
Specifies the *S*CCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

–e Requests information for all deltas created *earlier* than and including the delta designated via the **–r** option.

–l Requests information for all deltas created *later* than and including the delta designated via the **–r** option.

–a Requests printing of information for both removed, that is, delta type = *R*, (see *rmdel*(1)) and existing, that is, delta type = *D*, deltas. If the **–a** option is not specified, information for existing deltas only is provided.

In the absence of the **–d** options, *prs* displays a default set of infomation consisting of: delta-type, release number and level number, date and time last changed, user-name of the person who changed the file, lines inserted, changed, and unchanged, the MR numbers, and the comments.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: 1) the user supplied text; and 2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by **\t** and carriage return/new-line is specified by **\n**.

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

```
* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:
```

EXAMPLES

/usr/sccs/prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

/usr/sccs/prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case:*

/usr/sccs/prs s.file

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only option argument allowed to be used with the *special case* is the **-a** option.

FILES

/tmp/pr?????

SEE ALSO

sccs(1), admin(1), delta(1), get(1), help(1), sccsfile(5).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps − process status

SYNOPSIS

ps [**acCegklsStuvwx#**] [*kernel_name*] [*c_dump_file*] [*swap_file*]

DESCRIPTION

Ps displays information about processes. Normally, only processes that you have started are candidates to be displayed by *ps*, but see the **OPTIONS** section below for how to get more information. Specifying **a** makes other users' processes candidates to be displayed; specifying **x** includes processes without control terminals in the candidate pool.

All output formats include, for each process, the process id — PID, control terminal of the process — TT, cpu time used by the process — TIME (this includes both user and system time), the state — STAT — of the process, and an indication of the COMMAND which is running.

The state is given by a sequence of four letters, for example, 'RWNA'.

First letter indicates the runnability of the process:

R Runnable processes,
T Stopped processes,
P Processes in page wait,
D Processes in disk (or other short term) waits,
S Processes sleeping for less than about 20 seconds,
I Processes which are idle (sleeping longer than about 20 seconds).
Z A child processes that has terminated and is waiting for its parent process to do a wait.

Second letter indicates whether a process is swapped out;

blank
(that is, a space) in this position indicates that the process is loaded (in memory).
W Process is swapped out.
> Process has specified a soft limit on memory requirements and has exceeded that limit shows; such a process is (necessarily) not swapped.

Third letter indicates whether a process is running with altered CPU scheduling priority (nice):

blank
(that is, a space) in this position indicates that the process is running without special treatment.
N The process priority is reduced,
< The process priority has been raised artificially.

Fourth letter indicates any special treatment of the process for virtual memory replacement. The letters correspond to options to the *vadvise*(2) system call. Currently the possibilities are:

blank
(that is, a space) in this position stands for VA_NORM.
A Stands for VA_ANOM. An A typically represents a program which is doing garbage collection.
S Stands for VA_SEQL. An S is typical of large image processing programs which are using virtual memory to sequentially address voluminous data.

Kernel_name specifies the location of the system namelist. If the **k** option is given, *c_dump_file* tells **ps** where to look for *core*. Otherwise, the core dump is located in the file */vmcore* and this argument is ignored. *Swap_file* gives the location of a swap file other than the default, */dev/drum*.

OPTIONS

a Display information about all processes with terminals (ordinarily only one's own processes are displayed).

c Display the command name, as stored internally in the system for purposes of accounting, rather than the command arguments, which are kept in the process' address space. This is more reliable, if less informative, since the process is free to destroy the latter information.

C Display raw CPU time in the %CPU field instead of the decaying average.

e Display the environment as well as the arguments to the command.

g Display all processes. Without this option, *ps* only prints 'interesting' processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to login on free terminals.

k Normally, *kernel_name*, defaults to */vmunix*, *c_dump_file* is ignored, and *swap_file* defaults to */dev/drum*. With the **k** option in effect, these arguments default to */vmunix*, */vmcore*, and */dev/drum*, respectively.

l Display a long listing, with fields PPID, CP, PRI, NI, ADDR, SIZE, RSS and WCHAN as described below.

s Adds the size SSIZ of the kernel stack of each process (for use by system maintainers) to the basic output format.

S Display accumulated CPU time used by this process and all of its reaped children.

t*x* Restrict output to processes whose controlling tty is *x* (which should be specified as printed by *ps,* for example, *t3* for tty3, *tco* for console, *td0* for ttyd0, *t?* for processes with no tty, etc). This option must be the last one given.

u Display user-oriented output. This includes fields USER, %CPU, NICE, SIZE, and RSS as described below.

v Display a version of the output containing virtual memory. This includes fields RE, SL, PAGEIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU and %MEM, described below.

w Use a wide output format (132 columns rather than 80); if repeated, that is, ww, use arbitrarily wide output. This information is used to decide how much of long commands to print.

x Display even those processes with no terminal.

A process number may be given, in which case the output is restricted to that process. This option must also be last.

DISPLAY FORMATS

Fields which are not common to all output formats:

USER name of the owner of the process

%CPU cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all %CPU fields to exceed 100%.

NICE (or NI) process scheduling increment (see *setpriority*(2) and *nice*(3C).

SIZE virtual size of the process (in 1024 byte units)

RSS real memory (resident set) size of the process (in 1024 byte units)

LIM soft limit on memory used, specified via a call to *getrlimit*(2); if no limit has been specified then shown as *xx*

TSIZ size of text (shared program) image

TRS size of resident (real memory) set of text

%MEM percentage of real memory used by this process.

RE residency time of the process (seconds in core)

SL sleep time of the process (seconds blocked)

PAGEIN number of disk i/o's resulting from references by the process to pages not loaded in core.

UID numerical user-id of process owner

PPID numerical id of parent of process

CP short-term cpu utilization factor (used in scheduling)

PRI process priority (non-positive when in non-interruptible wait)

ADDR swap address of the process

WCHAN event on which process is waiting (an address in the system), with the initial part of the address trimmed off, for example, 80004000 prints as 4000.

F flags associated with process as in <*sys/proc.h*>:

SLOAD	0000001	in core
SSYS	0000002	swapper or pager process
SLOCK	0000004	process being swapped out
SSWAP	0000008	save area flag
STRC	0000010	process is being traced
SWTED	0000020	another tracing flag
SULOCK	0000040	user settable lock in core
SPAGE	0000080	process in page wait state
SKEEP	0000100	another flag to prevent swap out
SOMASK	0000200	restore old mask after taking signal
SWEXIT	0000400	working on exiting
SPHYSIO	0000800	doing physical i/o (bio.c)
SVFORK	0001000	process resulted from vfork()
SVFDONE	0002000	another vfork flag
SNOVM	0004000	no vm, parent in a vfork()
SPAGI	0008000	init data space on demand, from inode
SSEQL	0010000	user warned of sequential vm behavior
SUANOM	0020000	user warned of anomalous vm behavior
STIMO	0040000	timing out during sleep
SOUSIG	0100000	using old signal mechanism
SOWEUPC	0200000	owe process an addupc() call at next ast
SSEL	0400000	selecting; wakeup/waiting danger
SLOGIN	0800000	a login process (legit child of init)
SPTECHG	1000000	pte's for process have changed

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>; a process which is blocked trying to exit is marked <exiting>; *ps* makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory
/dev/drum	swap device
/vmcore	core file
/dev	searched to find swap device and tty names

SEE ALSO

kill(1), w(1)

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality.

NAME

pti – phototypesetter interpreter

SYNOPSIS

pti [file **...**]

DESCRIPTION

Pti shows the commands in a stream from the standard output of *troff*(1) using *troff's* **–t** option, interpreting them as they would act on the typesetter. Horizontal motions shows as counts in internal units and are marked with '<' and '>' indicating left and right motion. Vertical space is called *leading* and is also indicated.

The output is really cryptic unless you are an experienced C/A/T hardware person. It is better to use **troff –a**.

SEE ALSO

troff(1)

NAME

ptx — permuted index

SYNOPSIS

ptx [**−f**] [**−t**] [**−w** *n*] [**−g** *n*] [**−o** *only*] [**−i** *ignore*]] **−b** *break*]] **−r**] [input [output]]

DESCRIPTION

Ptx generates a permuted index of the contents of file *input* onto file *output* (defaults are standard input and output). *Ptx* has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

```
.xx "tail" "before keyword" "keyword and after" "head"
```

where .xx may be an *nroff*(1) or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

OPTIONS

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter; the default line length is 100 characters.

−w*n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

−g*n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

−o*only* Use as keywords only the words given in the *only* file.

−i*ignore*
Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use */usr/lib/eign* as the *ignore* file.

−b*break*
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

−r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

FILES

/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

pwd − print working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

If you are using *csh*(1), you can use the *dirs* builtin command to do the same job more quickly; *BUT dirs* can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it. This is because *pwd* searches back up the directory tree to report the true pathname, whereas *dirs* remembers the pathname from the last *cd* command. The example below illustrates the differences.

```
% cd  /usr/wendy/january/reports
% pwd
/usr/wendy/january/reports
% dirs
~/january/reports
% mv   ~/january   ~/february
% pwd
/usr/wendy/february/reports
% dirs
~/january/reports
%
```

Pwd and *dirs* also give different answers when you change directory through a symbolic link. For example:

```
% cd  /usr/wendy/january/reports
% pwd
/usr/wendy/january/reports
% dirs
~/january/reports
% ls −l /usr/wendy/january
lrwxrwxrwx  1 wendy         17 Jan 30  1983 /usr/wendy/january -> /usr/wendy/1984/jan/
% cd  /usr/wendy/january
% pwd
/usr/wendy/1984/jan
% dirs
/usr/wendy/january
```

They may also report different pathnames if you have changed directories through a symbolic link.

SEE ALSO

cd(1), csh(1), getwd(3)

NAME

px – Pascal interpreter

SYNOPSIS

px [obj [argument ...]]

DESCRIPTION

Px interprets the abstract machine code generated by *pi*. The first argument is the file to be interpreted, and defaults to *obj*; remaining arguments are available to the Pascal program using the built-ins *argv* and *argc*. *Px* is also invoked by *pix* when running 'load and go'.

If the program terminates abnormally an error message and a control flow backtrace are printed. The number of statements executed and total execution time are printed after normal termination. The **p** option of *pi* suppresses all of this except the message indicating the cause of abnormal termination.

FILES

obj	default object file
pmon.out	profile data file

SEE ALSO

The *Pascal User's Manual* in the Sun *Pascal* Manual.
pi(1), pix(1)

DIAGNOSTICS

Most run-time error messages are self-explanatory. Some of the more unusual ones are:

Reference to an inactive file

A file other than *input* or *output* was used before a call to *reset* or *rewrite*.

Statement count limit exceeded

The limit of 500,000 executed statements (which prevents excessive looping or recursion) has been exceeded.

Bad data found on integer read
Bad data found on real read

Usually, non-numeric input was found for a number. For reals, Pascal requires digits before and after the decimal point so that numbers like '.1' or '21.' evoke the second diagnostic.

panic: *Some message*

Indicates a internal inconsistency detected in *px* probably due to a Pascal system bug.

BUGS

Post-mortem traceback is not limited; infinite recursion leads to almost infinite traceback.

NAME

pxp − Pascal execution profiler

SYNOPSIS

pxp [**−acdefjLnstuw_**] [**−23456789**] [**−z** [name ...]] name.p

DESCRIPTION

Pxp can be used to obtain execution profiles of Pascal programs or as a pretty-printer. To produce an execution profile all that is necessary is to translate the program specifying the **z** option to *pc*, *pi*, or *pix*, execute the program, and then type the command

```
tutorial% pxp −z name.p
```

Pxp generates a reformatted listing if none of the **c**, **t**, or **z** options are specified; thus

```
tutorial% pxp old.p > new.p
```

places a pretty-printed version of the program in *old.p* in the file *new.p*.

OPTIONS

The use of the following options of *pxp* is discussed in the *Pascal User's Manual* in the Sun *Pascal* Manual.

−a Print the bodies of all procedures and functions in the profile; even those which were never executed.

−c Extract profile data from the file *core*.

−d Include declaration parts in a profile.

−e Eliminate **include** directives when reformatting a file; the **include** is replaced by the reformatted contents of the specified file.

−f Fully parenthesize expressions.

−j Left justify all procedures and functions.

−L Map all identifiers and keywords to lower case.

−n Eject a new page as each file is included; in profiles, print a blank line at the top of the page.

−s Strip comments from the input text.

−t Print a table summarizing **procedure** and **function** call counts.

−u Card image mode; only the first 72 characters of input lines are used.

−w Suppress warning diagnostics.

−z Generate an execution profile. If no *names* are given the profile is of the entire program. If a list of *names* is given, then only the specified **procedures** or **functions** and the contents of the specified **include** files will appear in the profile.

−_ Underline keywords.

−*d* Use *d* spaces (where *d* is a digit, $2 \le d \le 9$) as the basic indenting unit. The default is 4.

FILES

name.p	input file
name.i	include file(s)
name.h	include file(s)
pmon.out	profile data
core	profile data source for **−c** option
/usr/lib/how_pxp	information on basic usage

SEE ALSO

The *Pascal User's Manual* in the Sun *Pascal* Manual.

pc(1), pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

```
tutorial% pxp
```

Error diagnostics include 'No profile data in file' with the **c** option if the **z** option was not enabled to *pi;* 'Not a Pascal system core file' if the core is not from a *px* execution; 'Program and count data do not correspond' if the program was changed after compilation, before profiling; or if the wrong program is specified.

BUGS

Does not place multiple statements per line.

Procedures and functions as parameters are printed without nested parameter lists, as in the obsolete Jensen and Wirth syntax.

NAME

pxref — Pascal cross-reference program

SYNOPSIS

pxref [—] name

DESCRIPTION

Pxref makes a line numbered listing and a cross-reference of identifier usage for the program in *name.* The optional '—' argument suppresses the listing. The keywords **goto** and **label** are treated as identifiers for the purpose of the cross-reference. **Include** directives are not processed, but cause the placement of an entry indexed by '#include' in the cross-reference.

SEE ALSO

The *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.

BUGS

Identifiers are trimmed to 10 characters.

NAME

ranlib – convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

Ranlib converts each *archive* to a form which the loader can load more rapidly. *Ranlib* does this by adding a table of contents called **__.SYMDEF** to the beginning of the archive. *Ranlib* uses *ar*(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system which contains the current directory.

SEE ALSO

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

BUGS

Because generation of a library by *ar* and randomization of the library by *ranlib* are separate processes, phase errors are possible. The loader, *ld*, warns when the modification date of a library is more recent than the creation date of its dictionary; but this means that you get the warning even if you only copy the library.

NAME

rastrepl – magnify a raster image by 2 times

SYNOPSIS

rastrepl [*input-file* [*output-file*]]

DESCRIPTION

Rastrepl reads *input-file* (or the standard input if *input-file* is not specified) which should be in rasterfile format (see */usr/include/rasterfile.h*), replicates each pixel in both the *x* and *y* directions, and writes the resulting rasterfile to *output-file* (or the standard output if *output-file* is not specified).

EXAMPLES

```
tutorial% screendump ¦ rastrepl ¦ lpr –Pversatec –v
```

sends a rasterfile containing the current frame buffer to the Versatec plotter, doubling the size of the image so that it fills a single page.

SEE ALSO

screendump(1), screenload(1), lpr(1)

NAME

ratfor – rational Fortran dialect

SYNOPSIS

ratfor [**−6** *c*] [**−C**] [**−h**] [filename **...**]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:

```
{ statement; statement; statement }
```

decision-making:

```
if (condition) statement [ else statement ]
switch (integer value) {
        case integer:     statement
        ...
        [ default: ]      statement
}
```

loops:

```
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [ until (condition) ]
break
next
```

and some syntactic sugar to make programs easier to read and write:

free form input:

multiple statements/line; automatic continuation

comments:

# this is a comment

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return(expression)

returns expression to caller from function

define: define name replacement

include:

include filename

Ratfor is best used with *f77*(1).

OPTIONS

−6 *c* Use the character *c* as the continuation character in column **6** when translating to Fortran. The default is to use the **&** character as a continuation character.

−C Pass Ratfor comments through to the translated code.

−h Translate Ratfor string constants to Hollerith constants of the form ***nnn*h** ***string***. Otherwise just pass the strings through to the translated code.

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

rcp – remote file copy

SYNOPSIS

rcp file1 file2
rcp [**−r**] file ... directory

DESCRIPTION

Rcp copies files between machines. Each ***file*** or ***directory*** argument is either a remote file name of the form "rhost:path", or a local file name (containing no ':' characters, or a '/' before any ':'s.)

If the **−r** is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

If ***path*** is not a full path name, it is interpreted relative to your login directory on ***rhost***. A ***path*** on a remote host may be quoted (using \, ", or ´) so that the metacharacters are interpreted remotely.

Rcp does not prompt for passwords; your current local user name must exist on ***rhost*** and allow remote command execution via *rsh*(1C).

Rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form "rhost.rname" to use ***rname*** rather than the current user name on the remote host.

Please note: *rcp* is meant to copy from one host to another; if by some chance you try to copy a file on top of itself, you will end up with a severely corrupted file (for example, if you executed the following command from host george: 'george% rcp testfile george:/usr/me/testfile'). Remember where you are at all times (putting your hostname in your prompt helps with this)!

SEE ALSO

ftp(1C), rsh(1C), rlogin(1C)

BUGS

Doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

Is confused by any output generated by commands in a .login, .profile, or .cshrc file on the remote host.

Rcp doesn't copy ownership, mode, and timestamps to the new files.

Rcp requires that the source host have permission to execute commands on the remote host when doing third-party copies.

NAME

readnews – read news articles

SYNOPSIS

readnews [**−a** *date*] [**−n** *newsgroups*] [**−t** *titles*] [**−lprxhfuM**] [**−c** [*mailer*]]

readnews −s

DESCRIPTION

Readnews without argument displays unread articles.

readnews −s displays the newsgroup subscription list.

Readnews maintains a *.newsrc* file in your home directory that specifies all news articles already read. *.newsrc* is updated at the end of each news reading session in which the **−x** or **−l** options weren't specified. If the NEWSRC environment variable is present, it should be the path name of a file to be used in place of *.newsrc*.

An options line may be placed in the *.newsrc* file. The options line starts with the word **options** (left justified) followed by the list of standard options just as they would be typed on the *readnews* command line. The list of options may include: the **−n** flag along with a newsgroup list; a favorite interface to use for reading the news; and/or the **−r** or **−t** flag. Continuation lines are specified by following lines beginning with a space or tab character. Similarly, options can be specified in the NEWSOPTS environment parameter. Options on the command line override options in the *.newsrc* file and options in the *.newsrc* file override options in the NEWSOPTS environment parameter.

When you use the reply command of the *mail*(1) or */bin/mail*(1) interfaces, *readnews* uses the MAILER environment parameter to determine which mailer to use. The default is usually *mail*.

You can specify a particular paging program for paging through articles. The PAGER environment parameter should be set to the name of the paging program. The name of the article is referenced with a '%', as in the **−c** option. If no '%' is present, the article is piped to the program. Paging may be disabled by setting PAGER to a null value.

OPTIONS

Some of the option flags determine which of the several interfaces you can use for reading your news. The news system has its own interface which is used if no other choice is made, otherwise one of these options can be used:

−M An interface to *mail*(1).

−c A */bin/mail*(1)–like interface.

−c *'mailer'*
All selected articles written to a temporary file. Then the mailer is invoked. The name of the temporary file is referenced with a '%'. Thus, 'mail −f %' will invoke mail on a temporary file consisting of all selected messages.

Other options govern the behavior of *readnews* itself, as follows:

−p Send all selected articles to the standard output with no questions asked.

−l Display only the titles. Do not update the *.newsrc* file.

−r display the articles in reverse order.

−f Do not display any followup article

−h Display articles in a less verbose format (intended for terminals running at 300 baud).

−u Update the *.newsrc* file every 5 minutes, in case of an unreliable system. Note that if the *.newsrc* file is updated, the **x** command will not restore it to its original contents.

The following flags determine the selection of articles:

−n *newsgroups*
: Select all articles that belong to *newsgroups*.

−t *titles*
: Select all articles whose titles contain one of the strings specified by *titles*.

−a [*date*]
: Select all articles that were posted past the given *date* (in *mm/dd* format).

−x Ignore *.newsrc* file. That is, select articles that have already been read as well as new ones.

COMMANDS

This section lists the commands you can type to the *readnews* and */bin/mail* interface prompts. The *readnews* interface suggests some common commands in brackets. Just typing carriage-return is the same as typing the first command. For example, '[ynq]' means that the commands 'y' (yes), 'n' (no), and 'q' (quit) are common responses, and that 'y' is the default. Here are the commands and their meanings:

y Yes — displays current article and goes on to next.

n No — goes on to next article without displaying current one. In the */bin/mail* interface, this means 'go on to the next article', which has the same effect as 'y' or just typing carriage-return.

q Quit — the *.newsrc* file is updated if −l or −x were not on the command line.

c Cancel the article — only the author or the super user can do this.

r Reply — reply to article's author via mail. *Readnews* calls up your EDITOR with a header specifying To, Subject, and References lines taken from the message. You may change or add headers, as appropriate. You add the text of the reply after the blank line, and then exit the editor. The resulting message is mailed to the author of the article.

rd Reply directly — *readnews* calls up the *mail* program (or the program specified in the $MAILER environment variable) so that you can reply to the author. Type the text of the reply and then control-D.

f [*title*] Submit a follow up article. Normally you should leave off the title, since the system generates one for you. *Readnews* calls up your EDITOR so that you can compose the text of the followup.

fd Followup directly, without edited headers. This is like *f*, but the headers of the article are not included in the editor buffer.

N [*newsgroup*]
: Go to the next newsgroup or named *newsgroup*.

s [*file*] Save — the article is appended to the named *file*. The default filename is *Articles*. If the first character of the filename is '|', the rest of the filename is taken as the name of a program, which is executed with the text of the article as standard input. If the first character of the filename is '/', it is taken as a full path name of a file. If the $NEWSBOX environment variable is set to a full path name, and the filename contains no '/', the file is saved in $NEWSBOX, otherwise, it is saved relative to $HOME.

# Report the name and size of the newsgroup.

e Erase — forget that this article was read.

h Print a more verbose header.

H Print a very verbose header, containing all known information about the article.

U Unsubscribe from this newsgroup. Also goes on to the next newsgroup.

d Read a digest. Breaks up a digest into separate articles so you can read and reply to each piece.

D Decrypt — invokes a Caesar decoding program on the body of the message. This is used to decrypt rotated jokes posted to net.jokes. Such jokes are usually obscene or otherwise offensive to some groups of people, and so are rotated to avoid accidental decryption by people who would be offended. The title of the joke should indicate the nature of the problem, enabling people to decide whether to decrypt it or not.

Normally the Caesar program does a character frequency count on each line of the article separately, so that lines which are not rotated will be shown in plain text. This works well unless the line is short, in which case it sometimes gets the wrong rotation. An explicit *number* rotation (usually 13) may be given to force a particular shift.

v Print the current version of the news software.

! Shell escape.

number
Go to *number*.

+[*n*] Skip n articles. The articles skipped are recorded as 'unread' and will be offered to you again the next time you read news.

– Go back to last article. This is a toggle, typing it twice returns you to the original article.

x Exit — like quit except that *.newsrc* is not updated.

X *system*
Transmit article to the named *system*.

The c, f, fd, r, rd, e, h, H, and s commands can be followed by –'s to refer to the previous article. Thus, when replying to an article using the *readnews* interface, you should normally type 'r–' (or 're-') since by the time you enter a command, you are being offered the next article.

EXAMPLES

readnews
Read all unread articles using the *readnews*(1) interface. The *.newsrc* file is updated at the end of the session.

readnews –c ed % –l
Use the *ed*(1) text editor on a file containing the titles of all unread articles. The *.newsrc* file is **not** updated at the end of the session.

readnews –n all !fa.all –M –r
Read all unread articles except articles whose newsgroups begin with IB fa . via *mail*(1) in reverse order. The *.newsrc* file is updated at the end of the session.

readnews –p –n all –a last thursday
Print every unread article since last Thursday. The *.newsrc* file is updated at the end of the session.

readnews –p > /dev/null &
Discard all unread news. This is useful after returning from a long trip.

FILES

/usr/spool/news/*newsgroup*/*number*	News articles
/usr/lib/news/active	Active newsgroups and numbers of articles
/usr/lib/news/help	Help file for *readnews*(1) interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

checknews(1), inews(1), sendnews(8), recnews(8), uurec(8), mail(1), news(5), newsrc(5)

NAME

recnews — receive unprocessed articles via mail

SYNOPSIS

/usr/lib/news/recnews [*newsgroup* [*sender*]]

DESCRIPTION

Recnews reads a letter from the standard input; determines the article title, sender, and newsgroup; and gives the body to inews with the right arguments for insertion.

If *newsgroup* is omitted, the to line of the letter is used. If *sender* is omitted, the sender is determined from the from line of the letter. The title is determined from the subject line.

SEE ALSO

inews(1), uurec(8), sendnews(8), readnews(1), checknews(1)

NAME

refer — find and insert literature references in documents

SYNOPSIS

refer [**−a***r*] [**−b**] [**−c***string*] [**−e**] [**−k***x*] [**−l***m,n*] [**−p** *file*] [**−n**] [**−s***keys*] file ...

DESCRIPTION

Refer is a preprocessor for *nroff*(1), or *troff*(1), that finds and formats references. The input files (standard input by default) are copied to the standard output, except for lines between **.**[and **.**] command lines. Such lines are assumed to contain keywords as for *lookbib*(1), and are replaced by information from a bibliographic data base. The user can avoid the search, override fields from it, or add new fields. The reference data, from whatever source, is assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. A flag is placed in the text at the point of reference. By default, the references are indicated by numbers.

When *refer* is used with *eqn*(1), *neqn*(1), or *tbl*(1), *refer* should be used first in the sequence, to minimize the volume of data passed through pipes.

OPTIONS

−a*r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted, all author names are reversed.

−b Bare mode — do not put any flags in text (neither numbers or labels).

−c*string*
Capitalize (with SMALL CAPS) the fields whose key-letters are in *string*.

−e Accumulate references instead of leaving the references where encountered, until a sequence of the form:

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

−k*x* Instead of numbering references, use labels as specified in a reference data line beginning with the characters *%x*; By default, *x* is **L**.

−l*m,n* Instead of numbering references, use labels from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either of *m* or *n* is omitted, the entire name or date, respectively, is used.

−p Take the next argument as a file of references to be searched. The default file is searched last.

−n Do not search the default file.

−s*keys* Sort references by fields whose key-letters are in the *keys* string, and permute reference numbers in the text accordingly. Using this option implies the **−e** option. The key-letters in *keys* may be followed by a number indicating how many such fields are used, with a + sign taken as a very large number. The default is **AD**, which sorts on the senior author and date. To sort on all authors and then the date, for instance, use the options **−sA+T**.

FILES

/usr/dict/papers	directory of default publication lists and indexes
/usr/lib/refer	directory of programs

NAME

rev — reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

NAME

rlogin — remote login

SYNOPSIS

rlogin *rhost* [**−e** *c*] [**−l** *username*] [**−8**]

rhost [**−e** *c*] [**−l** *username*] [**−8**]

DESCRIPTION

Rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rsh*(1C).) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file *.rhosts* in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user and host is not found in these files, the remote machine will prompt for a password as in *login*(1). To avoid some security problems, the *.rhosts* file must be owned by either the remote user or root and may not be a symbolic link.

Your remote terminal type is the same as your local terminal type (as given in your environment TERM variable). All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly.

ESCAPES

Lines starting with the tilde character are 'escape sequences' (the escape character can be changed via the **−e** options):

~. Disconnect from the remote host — this is not the same as a logout, because the local host breaks the connection with no warning to the remote end.

~^Z Suspend the login session (only if you are using the C-Shell). *Susp* is your 'suspend' character — usually control-Z — see *tty*(1).

~^Y Suspend the input half of the login, but output will still be seen (only if you are using the C-Shell). *Dsusp* is your 'deferred suspend' character — usually control-Y — see *tty*(1).

OPTIONS

−l Specifies a different user name (*username*, in the synopsis) for the remote login. If you do not use this option, the remote username used is the same as your local username.

−e Specifies a different escape character (*c*, in the synopsis) for the line used to disconnect from the remote host.

−8 Pass eight-bit data across the net instead of seven-bit data.

SEE ALSO

rsh(1C), stty(1)

FILES

/usr/hosts/*	for *rhost* version of the command
/etc/hosts	to translate hostname to network address
/etc/hosts.equiv	list of *rhosts* with shared account names
~/.rhosts	private list of *rhosts* with shared account names
/etc/services	to translate service name tcp/rlogin to network port number

BUGS

This implementation can only use the TCP network service.

More terminal characteristics should be propagated.

NAME

rm, rmdir – remove (unlink) files or directories

SYNOPSIS

rm [**−f**] [**−r**] [**−i**] [**−**] file ...

rmdir dir ...

DESCRIPTION

rm.1:.IX "rm command" "" "**rm** — remove file or directory" rm.1:.IX "remove file or directory" "" "remove file or directory — **rm**" rm.1:.IX "delete file or directory" "" "delete file or directory — **rm**" *Rm* removes the directory entries for one or more files. If an entry was the last link to the file, the file is destroyed. *Rm* **−r** and *rmdir* remove entries for directories.

To remove a file, you must have write permission in its directory; but you don't need read or write permission on the file itself. If you don't have write permission on the file and the standard input is a terminal, *rm* displays the file's permissions and waits for you to type in a response. If your response begins with 'y' the file is deleted; otherwise the file is left alone.

To remove a full directory, use *rm* with the **−r** option (see below). *Rmdir* removes the named directory only if it is empty.

OPTIONS

The following are options for *rm*:

−f Force files to be removed, without displaying permissions, asking questions, or reporting errors.

−r **R**ecursively delete the entire contents of the specified directory and the directory itself.

−i Ask whether to delete each file, or, under **−r**, whether to examine each directory. Sometimes called the **i**nteractive option.

− Treat all the following arguments as filenames — so that you can specify filenames starting with a minus.

WARNING

It is forbidden to remove the file '..' merely to avoid the antisocial consequences of inadvertently doing something like 'rm −r .*'.

SEE ALSO

rmdir(1), rmdir(2), unlink(2)

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

/usr/sccs/rmdel **–r**SID file ...

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS *file*. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (that is, if a *p-file* (see *get*(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*).

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

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either 1) if you make a delta you can remove it; or 2) if you own the file and directory you can remove a delta.

FILES

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

SEE ALSO

sccs(1), delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System in *Programming Tools for the Sun Workstation*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

rmdir, rm – remove (unlink) directories or files

SYNOPSIS

rmdir dir ...

rm [**–f**] [**–r**] [**–i**] [–] file ...

DESCRIPTION

Rmdir removes entries for the named directories, which must be empty.

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked and no errors are reported when the **–f** (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument **–r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **–i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **–r**, whether to examine each directory.

The null option – indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

SEE ALSO

rm(1), unlink(2), rmdir(2)

NAME

rsh – remote shell

SYNOPSIS

rsh host [**–l** username] [**–n**] command

host [**–l** username] [**–n**] command

DESCRIPTION

Rsh connects to the specified *host,* and executes the specified *command. Rsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does.

If you omit *command*, instead of executing a single command, *rsh* logs you in on the remote host using *rlogin*(1C).

Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command:

```
        tutorial% rsh lizard cat lizard.file >> tutorial.file
```

appends the remote file *lizard.file* from the machine called *lizard* to the file called *tutorial.file* on the machine called *tutorial*, while the command:

```
        tutorial% rsh lizard cat lizard.file ">>" another.lizard.file
```

appends the file *lizard.file* on the machine called *lizard* to the file *another.lizard.file*. which also resides on the machine called *lizard*.

Host names are given in the file /etc/hosts. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory /usr/hosts; if you put this directory in your search path then the **rsh** can be omitted.

OPTIONS

–l *username*

use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username, This remote name must be equivalent (in the sense of *rlogin*(1C)) to the originating account; no provision is made for specifying a password with a command.

–n redirect the input of *rsh* to */dev/null*. You need this option if you are using *csh*(1) and put a *rsh*(1C) in the background without redirecting its input away from the terminal because it will block even if no reads are posted by the remote command.

The type of remote shell (sh or csh) is determined by the user's entry in the file */etc/passwd* on the remote system.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1C)

BUGS

You cannot run an interactive command (like *vi*(1)); use *rlogin*(1C).

Stop signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

The current local environment is not passed to the remote shell.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**−a**] [**−l**] [**−t**] [**−u**]

DESCRIPTION

Ruptime gives a status line like *uptime* for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name, but this order can be changed by specifying one of the options listed below.

OPTIONS

−a
count even those users who have been idle for an hour or more.

−l sort the display by load average.

−t
sort the display by up time.

−u
sort the display by number of users.

FILES

/usr/spool/rwho/whod.*	data files

SEE ALSO

rwho(1C)

NAME

rwall — write to all users over a network

SYNOPSIS

rwall host1 host2 ...
rwall **—n** netgroup1 netgroup2 ...
rwall **—h** host **—n** netgroup

DESCRIPTION

Rwall reads a message from standard input until end-of-file. It then sends this message, preceded by the line "Broadcast Message ...", to all users logged in on the specified host machines. With the **-n** option, it sends to the specified network groups, which are defined in *netgroup*(5).

A machine can only receive such a message if it is running *rwalld*(8), which is normally started up from */etc/servers* by the daemon *inetd*(8).

FILES

/etc/servers

SEE ALSO

wall(1), netgroup(5), rwalld(8), shutdown(8)

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [**−a**]

DESCRIPTION

The *rwho* command produces output similar to *who*(1), but for all machines on your network. If no report has been received from a machine for 5 minutes, *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, the user is omitted from the output of *rwho* unless the **−a** flag is given.

OPTIONS

−a report all users whether or not they have typed to the system in the past hour.

FILES

/usr/spool/rwho/whod.*	information about other machines

SEE ALSO

ruptime(1C), rwhod(8C)

BUGS

Does not work through gateways.

This is unwieldy when the number of machines on the local net is large.

NAME

sact − print current SCCS file editing activity

SYNOPSIS

/usr/sccs/sact file ...

DESCRIPTION

Sact informs the user of any SCCS files which have had one or more **get −e** commands applied to them, that is, there are files out for editing, and deltas are pending. If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. The output for each named file consists of five fields separated by spaces.

Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 specifies the SID for the new delta to be created.

Field 3 contains the logname of the user who will make the delta (that is, executed a *get* for editing).

Field 4 contains the date that **get −e** was executed.

Field 5 contains the time that **get −e** was executed.

SEE ALSO

sccs(1), delta(1), get(1), unget(1).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sccs – front end for the SCCS subsystem

SYNOPSIS

sccs [**–r**] [**–d***prefixpath*] [**–p***finalpath*] *command* [SCCS-flags ...] [file ...]

DESCRIPTION

system" The *sccs* command is a front end to the utility programs of the Source Code Control System (SCCS), which helps them mesh more cleanly with the rest of UNIX. *Sccs* runs the *command* with the specified SCCS-flags and *files*.

Note that *sccs* normally prefixes each *file*, or the last component of each *file*, with the string *SCCS/s.*, because you normally keep your SCCS database files in a directory called SCCS, and each database file starts with an *s.* prefix.

Sccs program options must appear before the *command* argument. Flags to be passed to the actual SCCS utility program must appear after the *command* argument. These flags are specific to the *command* being used, and are discussed in the manual page for that *command*.

Sccs also includes the capability to run 'set user id' to another user to provide additional protection. Certain commands (such as *admin*) cannot be run 'set user id' by all users, since this would allow anyone to change the authorizations. Such commands are always run as the real user.

OPTIONS

–r Runs *sccs* as the real user rather than as whatever effective user *sccs* is 'set user id' to.

–d*prefixpath*

Defines the prefix portion of the pathname for the SCCS database files. The default prefix portion of the pathname is the current directory. *Prefixpath* is prefixed to the entire pathname. For example:

```
    tutorial% sccs –d/usr/include get sys/inode.h
```

converts to:

```
    get /usr/include/sys/SCCS/s.inode.h
```

The intent here is to create aliases such as:

```
    alias syssccs sccs –d/usr/src
```

which will be used as:

```
    tutorial% syssccs get cmd/who.c
```

–p*finalpath*

Defines the name of a lower directory in which the SCCS files will be found; SCCS is the default. *Finalpath* is appended before the final component of the pathname. For example:

```
    tutorial% sccs –pprivate get usr/include/stdio.h
```

converts to:

```
    get usr/include/private/s.stdio.h
```

ADDITIONAL SCCS COMMANDS

Several 'pseudo-commands' are available in addition to the usual SCCS commands. These are:

admin The *admin* command is similar to that of the raw SCCS *admin* command. When creating new *s.* files, the *create* command (described just below) does more of the startup work for you and should be used in preference to *admin*.

create *Create* is used when creating new *s.* files. Given a C source language file called *obscure.c*, *Create* does the following actions: (1) creates the *s.* file called *s.obscure.c* in the SCCS directory; (2) renames the original source file to *,obscure.c;* (3) does an **sccs get** on *obscure.c*.

edit Get a file for editing.

delget Perform a *delta* on the named files and then *get* new versions. The new versions have id keywords expanded, and so cannot be edited.

deledit Same as *delget*, but produces new versions suitable for editing. *Deledit* is useful for making a 'checkpoint' of your current editing phase.

fix Removes the named delta, but leaves you with a copy of the delta with the changes that were in it. *Fix* must be followed by a **−r** flag. *Fix* is useful for fixing small compiler bugs, etc. Since *fix* doesn't leave audit trails, use it carefully.

clean Removes everything from the current directory that can be recreated from SCCS files. *Clean* checks for and does not remove any files being edited. If *Clean* **−b** is used, branches are **not** checked to see if they are currently being edited. Note that **−b** is dangerous if you are keeping the branches in the same directory.

unedit 'Undoes' the last *edit* or *get* **−e** and returns a file to its previous condition. If you *unedit* a file being edited, all changes made since the beginning of the editing session are lost.

info Displays a list of all files being edited. If the **−b** flag is given, branches (that is, SID's with two or fewer components) are ignored. If the **−u** flag is given (with an optional argument), only files being edited by you (or the named user) are listed.

check Checks for files currently being edited, like *info*, but returns an exit code rather than a listing: nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. *Check* may thus be included in an 'install' entry in a makefile, to ensure that everything is included in an SCCS file before a version is installed.

tell Displays a list of files being edited on the standard output. Filenames are separated by newlines. Takes the **−b** and **−u** flags like *info* and *check*.

diffs Compares (in *diff*-like format) the current version of the program you have out for editing and the versions in SCCS format.

EXAMPLES

To put a file called *myprogram.c* into SCCS format for the first time, assuming also that there is no SCCS directory already existing:

```
tutorial% mkdir SCCS
tutorial% sccs create myprogram.c

myprogram.c:
1.1
14 lines
   after you have verified that everything is all right
   you remove the version of the file that starts with a comma:
tutorial% rm ,myprogram.c
tutorial%
```

To get a copy of *myprogram.c* for editing, edit that file, then place it back in the SCCS database:

```
tutorial% sccs edit myprogram.c
1.1
new delta 1.2
14 lines
tutorial% vi myprogram.c
   your editing session
tutorial% sccs delget myprogram.c
comments? Added abusive responses for compatibility with rest of system
1.2
7 inserted
7 deleted
```

```
7 unchanged
1.2
14 lines
tutorial%
```

To *get* a file from another directory:

```
tutorial% sccs -p/usr/src/sccs/ get cc.c
```

or:

```
tutorial% sccs get /usr/src/sccs/cc.c
```

To make a delta of a large number of files in the current directory:

```
tutorial% sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
tutorial% sccs info -b
```

To *delta* everything that you are editing:

```
tutorial% sccs delta `sccs tell -u`
```

In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
        sccs get $(REL) $@
```

REGULAR SCCS COMMANDS

The 'regular' SCCS commands are described very briefly below. It is unlikely that you ever need to use these commands because the user interface is so complicated, and the *sccs* front end command does 99.9% of the interesting tasks for you.

admin Creates new SCCS files and changes parameters of exitsing SCCS files. You can use **sccs create** to create new SCCS files, or use **sccs admin** to do other things.

cdc Changes the commentary material in an SCCS delta.

comb Combines SCCS deltas and reconstructs the SCCS files.

delta Permanently introduces changes that were made to a file previously retrieved via **sccs get**. You can use **sccs delget** as the more useful version of this command since **sccs delget** does all of the useful work and more to boot.

get Extracts a file from the SCCS database, either for compilation, or for editing when the **-e** option is used. Use **sccs get** if you really need it, but **sccs delget** will normally have done this job for you. Use **sccs edit** instead of **get** with the **-e** option.

help Supposed to help you interpret SCCS error messages, but usually just parrots the messaage and is generally not considered very helpful.

prs Displays information about what is happening in an SCCS file.

rmdel Removes a delta from an SCCS file.

sact Displays information about editing activity in an SCCS file. The **sccs info** command is a lot more useful for the real life user.

sccsdiff Compares two versions of an SCCS file and generates the differences between the two versions. The **sccs delget** command does all this work as part of its normal process.

unget Undoes the work of a previous **get -e** command or a **sccs edit** command. Use the **sccs unedit** command as a useful alternative.

val Determines if a given SCCS file meets specified criteria. If you use the *sccs* command, you shouldn't need to use *val*, because its user interface is unbelievable.

what Displays SCCS identification information.

SEE ALSO

admin(1), comb(1), cdc(1), delta(1), get(1), help(1), prs(1), rmdel(1), sact(1), sccsdiff(1), sccsfile(5), unget(1), val(1), what(1)
Source Code Control System in *Programming Tools for the Sun Workstation*.

NAME

sccsdiff — compare two versions of an SCCS file

SYNOPSIS

/usr/sccs/sccsdiff —r *SID1* **—r** *SID2* [**—p**] [**—s** *n*] files

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but options apply to all files.

OPTIONS

—r*SID?* *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *diff*(1) in the order given.

—p pipe output for each file through *pr*(1).

—s*n* *n* is the file segment size that *diff* will use. This is useful when the system load is high.

FILES

/tmp/get????? Temporary files

SEE ALSO

sccs(1), diff(1), get(1), help(1), pr(1).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

"*file*: No differences" If the two versions are the same.
Use *help*(1) for explanations.

NAME

screendump – dump frame buffer image

SYNOPSIS

screendump [**–f** *display*] [**–e**] [**–t** *type*] [*output-file*]

DESCRIPTION

Screendump reads out the contents of a frame buffer on any model of Sun Workstation and outputs the display image in Sun standard rasterfile format (see */usr/include/rasterfile.h*) on *output-file* (or on the standard output if *output-file* is not specified).

By default, *screendump* attempts to output the contents of the console frame buffer (*/dev/fb*). The **–f** option explicitly sets the name of the desired frame buffer device.

The utility program *screenload* displays Sun standard rasterfiles on an appropriate Sun Workstation monitor. Output filters exist for printing standard monochrome rasterfiles on Versatec V-80 electrostatic plotters.

OPTIONS

–f *display*

Use *display* as the device name of the display.

-e Shorthand for **–t 2**.

–t *type*

Specify the type of the rasterfile to write:

0	Old format files compatible with Release 1.1 software.
1	Standard format.
2	Run-length encoding of bytes.
65535	To experiment with private encodings.

EXAMPLES

```
tutorial% screendump >save.this.image
```

writes the current contents of the console frame buffer into the file *save.this.image.*

```
tutorial% screendump –f /dev/cgone0 >save.color.image
```

writes the current contents of the frame buffer */dev/cgone0* into the file *save.color.image.*

```
tutorial% screendump ¦ lpr –Pversatec –v
```

sends a rasterfile containing the current frame buffer to the lineprinter, selecting the printer named "versatec" and the "v" output filter (see */etc/printcap*).

FILES

/dev/fb	Default name of console display frame buffer.
/dev/cgone0	Default name of the Sun-1 color display frame buffer.
/dev/cgtwo0	Default name of the Sun-2 color display frame buffer.
/usr/lib/rasfilters/*	Filters for the raster file

SEE ALSO

screenload(1), lpr(1)
pr_dump in *Programmer's Reference Manual for SunWindows*

NAME

screenload – restore frame buffer image

SYNOPSIS

screenload [**–d**] [**–f** *display*] [**–i***index*] [**–b**] [**–g**] [**–p**] [**–w**] [**–h#** [#### [...]]] [*input-file*]

DESCRIPTION

Screenload accepts input in Sun standard rasterfile format (see */usr/include/rasterfile.h*) and attempts to display the input on the appropriate monitor (monochrome or color). *Screenload* normally displays rasterfiles on the console display, but displays them on some heuristically determined color display if it finds no console display. *Screenload* displays color rasterfiles only on a color monitor. *Screenload* is able to display monochrome rasters on a color display.

If the dimensions of the image contained in the input data are smaller than the actual resolution of the display (for example, loading a 1024-by-800 Sun-1 image on a 1152-by-900 Sun-2 display), *screenload* centers the image on the actual workstation screen and fills the border area with solid black (by default). Various options may be used to change the fill pattern.

If the dimensions of the image contained in the input data are larger than the actual resolution of the display, *screenload* clips the right and bottom edges of the input image.

If there is an optional filename argument, *screenload* reads its input data from that file. If no filename argument is given, *screenload* reads its input data from the standard input.

The utility program *screendump* creates Sun standard rasterfiles from the display on a Sun Workstation monitor.

OPTIONS

–d Verify that display dimensions and input data image dimensions match, and print warning messages if they do not.

–f *display*
Use *display* as the name of the frame buffer device.

–i*index*
If the image is smaller than the display, use *index* (0 <= *index* < 256) as the index value into the color map for the foreground color of the border fill pattern. The default index value is 255.

–b If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of solid ones (*default*). On a monochrome display this results in a black border; on a color display the color map value selected by the **–i** option determines the border color.

–g If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of "desktop grey". On a monochrome display this results in a border matching the background pattern used by the SunWindows system; on a color display the color map value selected by the **–i** option determines the foreground border color, though the pattern is the same as on a monochrome display.

–p Wait for a newline to be typed on the standard input before exiting.

–w If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of solid zeros. On a monochrome display this results in a white border; on a color display the color map value at index 0 determines the border color.

–h# If the input data dimensions are smaller than the display dimensions, fill the border with the bit pattern described by the following # 16-bit hexadecimal constants. Note that a "1" bit is black and a "0" bit is white on the monochrome display; on a color diplay the color map value selected by the **–i** option determines the border foreground color. The number of hex constants in the pattern is limited to 16.

EXAMPLES

tutorial% **screenload saved.display.image**

reloads the raster image contained in the file *saved.display.image* on the display type indicated by the rasterfile header in that file.

tutorial% **screenload —f/dev/cgone0 monochrome.image**

reloads the raster image in the file *monochrome.image* on the frame buffer device */dev/cgone0.*

tutorial% **screenload —h1 ffff sun_1.saved.image**

is equivalent to the **—b** option (fill border with black), while

tutorial% **screenload —h4 8888 8888 2222 2222 sun_1.saved.image**

is equivalent to the **—g** option (fill border with desktop grey).

FILES

/dev/fb	Default name of console display frame buffer
/dev/cgone0	Default name of Sun-1 color display frame buffer
/dev/cgtwo0	Default name of the Sun-2 color display frame buffer.
/usr/lib/rasfilters/*	Filters for the raster file

SEE ALSO

screendump(1)
pr_load in *Programmer's Reference Manual for SunWindows*

NAME

script – make typescript of terminal session

SYNOPSIS

script [**–a**] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the **–a** option is given. It can be sent to the line printer later with *lpr*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

OPTIONS

Append the script to the specified file instead of writing over it.

BUGS

Script places **everything** in the log file. This is not what the naive user expects.

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** *script*] [**–f** *sfile*] [file] **...**

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands.

OPTIONS

–e Is a list of edit commands for *sed.* If there is just one **–e** option and no **–f**'s, the **–e** flag **–e** may be omitted.

–f Take the script from *sfile*

–n suppress the default output.

SED SCRIPTS

sed scripts consist of editing commands, one per line, of the following form:

> [address [, address]] function [arguments]

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a 'D' command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

An *address* is either a decimal number that counts input lines cumulatively across files, a '$' that addresses the last line of input, or a context address, '/regular expression/', in the style of *ed*(1) modified thus:

> The escape sequence '\n' matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. If the second address is a number less than or equal to the line number first selected, only one line is selected. Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function '!' (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

An argument denoted *text* consists of one or more lines, all but the last of which end with '\' to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an 's' command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An argument denoted *rfile* or *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most **10** distinct *wfile* arguments.

(1) a\
text

Append: Place *text* on the output before reading the next input line.

(2) b *label*

Branch to the ':' command bearing the *label.* Branch to the end of the script if *label* is empty.

(2)c\
text

Change: Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2)D Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2)g Replace the contents of the pattern space by the contents of the hold space.

(2)G Append the contents of the hold space to the pattern space.

(2)h Replace the contents of the hold space by the contents of the pattern space.

(2)H Append the contents of the pattern space to the hold space.

(1)i\
text

Insert: Place *text* on the standard output.

(2)n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2)N Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2)p Print: Copy the pattern space to the standard output.

(2)P Copy the initial segment of the pattern space through the first newline to the standard output.

(1)q Quit: Branch to the end of the script. Do not start a new cycle.

(2)r *rfile*

Read the contents of *rfile*. Place them on the output before reading the next input line.

(2)s/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of '/'. For a fuller description see *ed*(1). *Flags* is zero or more of

g Global: Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write: Append the pattern space to *wfile* if a replacement was made.

(2)t *label*

Test: Branch to the ':' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a 't'. If *label* is empty, branch to the end of the script.

(2)w *wfile*

Write: Append the pattern space to *wfile*.

(2)x Exchange the contents of the pattern and hold spaces.

(2)y/*string1*/*string2*/

Transform: Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*

Don't: Apply the *function* (or group, if *function* is '{') only to lines *not* selected by the address(es).

(0): *label*
This command does nothing; it bears a *label* for 'b' and 't' commands to branch to. Note that the maximum length of *label* is seven characters.

(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

ed(1), grep(1), awk(1), lex(1)
Using sed, the Stream Text Editor in *Editing and Text Processing on the Sun Workstation.*

NAME

sh, for, case, if, while, **:**, **.**, break, continue, cd, eval, exec, exit, export, login, newgrp, read, readonly, set, shift, times, trap, umask, wait — command language

SYNOPSIS

sh [**−ceiknrstuvx**] [arg] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See **Invocation** for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non blank *words* separated by blanks (a blank is a **tab** or a **space**). The first word specifies the name of the command to be executed. Except as specified below the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *execve*(2)). The *value* of a simple-command is its exit status if it terminates normally or 200+*status* if it terminates abnormally (see *sigvec*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more *pipelines* separated by **;**, **&**, **&&** or **||** and optionally terminated by **;** or **&**. **;** and **&** have equal precedence which is lower than that of **&&** and **||**; **&&** and **||** also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol **&&** causes the *list* following to be executed only if the preceding *pipeline* returns a zero value. The symbol **||** causes the *list* following to be executed only if the preceding *pipeline* returns a non zero value. Newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. The value returned by a command is that of the last simple-command executed in the command. Note that many keywords such as **done** are only recognized when they are the first keyword on a line.

```
for name [ in word ... ]
do
list
done
```

Each time a **for** command is executed *name* is set to the next word in the **in** *word* list. If **in** *word...* is omitted, **in $@** is assumed. Execution ends when there are no more words in the list.

```
case word in
[ pattern [ | pattern ] ... ) list ;; ]
[ pattern [ | pattern ] ... ) list ;; ]
...
esac
```

A **case** command executes the *list* associated with the first pattern that matches *word*. The form of the patterns is the same as that used for file name generation.

```
if list
then list
[elif list
then list ] ...
[ else
list ]
fi
```

The *list* following **if** is executed and if it returns zero the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero the *list* following **then** is executed. Failing that the **else** *list* is executed.

while *list*
[**do**
list]
done

A **while** command repeatedly executes the **while** *list* and if its value is zero executes the **do** *list;* otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list.* **until** may be used in place of **while** to negate the loop termination test.

(*list* **)** Execute *list* in a subshell.

{ *list* **}** *list* is simply executed.

The following words are only recognized as the first word of a command and when not quoted.

if then else elif fi case esac for while until do done { }

Command substitution.
The standard output from a command enclosed in a pair of back quotes (\` \`) may be used as part or all of a word; trailing newlines are removed.

Parameter substitution.
The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set**. Variables may be set by writing

name=value [*name=value*] ...

$**{***parameter***}**
A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters *** @ # ? − $!.** The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is a digit, it is a positional parameter. If *parameter* is ***** or **@** then all the positional parameters, starting with **$1**, are substituted separated by spaces. **$0** is set from argument zero when the shell is invoked.

$**{***parameter−word***}**
If *parameter* is set, substitute its value; otherwise substitute *word.*

$**{***parameter=word***}**
If *parameter* is not set, set it to *word;* the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

$**{***parameter?word***}**
If *parameter* is set, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

$**{***parameter+word***}**
If *parameter* is set, substitute *word;* otherwise substitute nothing.

In the above *word* is not evaluated unless it is to be used as the substituted string. (So that, for example, echo ${d−\`pwd\`} will only execute *pwd* if *d* is unset.)

The following *parameters* are automatically set by the shell.

#	The number of positional parameters in decimal.
−	Options supplied to the shell on invocation or by **set**.
?	The value returned by the last executed command in decimal.
$	The process number of this shell.

! The process number of the last background command invoked.

The following *parameters* are used but not set by the shell.

HOME The default argument (home directory) for the **cd** command.
PATH The search path for commands (see **execution**).
MAIL If this variable is set to the name of a mail file, the shell informs the user of the arrival of mail in the specified file.
PS1 Primary prompt string, by default '**$** '.
PS2 Secondary prompt string, by default '**>** '.
IFS Internal field separators, normally **space**, **tab**, and **newline**.

Blank interpretation.
After parameter and command substitution, any results of substitution are scanned for internal field separator characters (those found in **$IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File name generation.
Following substitution, each command word is scanned for the characters *****, **?** and **[**. If one of these characters appears, the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character **.** at the start of a file name or immediately following a **/**, and the character **/**, must be matched explicitly.

***** Matches any string, including the null string.
? Matches any single character.
[...] Matches any one of the characters enclosed. A pair of characters separated by — matches any character lexically between the pair.

Quoting.
The following characters have a special meaning to the shell and cause termination of a word unless quoted.

; & ( ) | < > newline space tab

A character may be *quoted* by preceding it with a backslash (\) character. **\\newline** is ignored. All characters enclosed between a pair of quote marks (´´), except a single quote, are quoted. Inside double quotes ("") parameter and command substitution occurs and \\ quotes the characters backslash (\), apostrophe (**'**), double quote (**"**), and dollar sign (**$**).

"**$***" is equivalent to "**$1 $2** ..." whereas
"**$@**" is equivalent to "**$1**" "**$2**" ...

Prompting.
When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (**$PS2**) is displayed.

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 (by seeking to the

end); otherwise the file is created.

<<*word*

The shell input is read up to a line the same as *word*, or end of file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\newline** is ignored, and **** is used to quote the characters backslash (****), dollar sign (**$**), apostrophe (**'**), and the first character of *word*.

<& *digit*

The standard input is duplicated from file descriptor *digit;* see *dup*(2). Similarly for the standard output using >.

<&− The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example,

```
... 2>&1
```

creates file descriptor 2 to be a duplicate of file descriptor 1.

If a command is followed by **&** then the default standard input for the command is the empty file (/dev/null). Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input output specifications.

Environment.

The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see *execve*(2) and *environ*(5). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a *parameter* for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these *parameters* or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's *parameter* to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to *parameters.* Thus these two lines are equivalent

```
TERM=450 cmd args
(export TERM; TERM=450; cmd args)
```

If the **−k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following prints 'a=b c' and 'c':

```
echo a=b c
set −k
echo a=b c
```

Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent. (But see also **trap.**)

Execution.

Each time a command is executed the above substitutions are carried out. Except for the 'special commands' listed below a new process is created and an attempt is made to execute the command via an *execve*(2).

The shell parameter **$PATH** defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (**:**). The default path is **:/bin:/usr/bin.**

If the command name contains a /, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an *a.out* file, it is assumed to be a file containing shell commands. A subshell (that is, a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands.
The following commands are executed in the shell process and except where specified no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. *file* Read and execute commands from *file* and return. The search path **$PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **$HOME** is the default *arg*.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input output arguments may appear and if no other arguments are given cause the shell input output to be modified.

exit [*n*]
Causes a non interactive shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. (An end of file will also exit from the shell.)

export [*name* ...]
The given names are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of exportable names is printed.

login [*arg* ...]
Equivalent to 'exec login arg ...'.

newgrp [*arg* ...]
Equivalent to 'exec newgrp arg ...'.

read *name* ...
One line is read from the standard input; successive words of the input are assigned to the variables *name* in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

readonly [*name* ...]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given, a list of all readonly names is printed.

set [**—eknptuvx** [*arg* ...]]

- **—e** If non interactive, exit immediately if a command fails.
- **—k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **—n** Read commands but do not execute them.
- **—t** Exit after reading and executing one command.
- **—u** Treat unset variables as an error when substituting.
- **—v** Print shell input lines as they are read.
- **—x** Print commands and their arguments as they are executed.
- **—** Turn off the **—x** and **—v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$—**.

Remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, etc. If no arguments are given, the values of all names are printed.

shift The positional parameters from **$2**... are renamed **$1**...

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
Arg is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by invoked commands. If *n* is 0, the command *arg* is executed on exit from the shell, otherwise upon receipt of signal *n* as numbered in *sigvec*(2). *Trap* with no arguments prints a list of commands associated with each signal number.

umask [*nnn*]
The user file creation mask is set to the octal value *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is that of the process waited for.

Invocation.
If the first character of argument zero is —, commands are read from **$HOME/.profile**, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

—c *string* If the **—c** flag is present, commands are read from *string*.

—s If the **—s** flag is present or if no arguments remain then commands are read from the standard input. Shell output is written to file descriptor 2.

—i If the **—i** flag is present or if the shell input and output are attached to a terminal (as told by *gtty*) then this shell is *interactive*. In this case the terminate signal SIGTERM (see *sigvec*(2)) is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptable). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command.

FILES

$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

Using the Bourne Shell in the *Beginner's Guide to the Sun Workstation*
csh(1), test(1), execve(2).

DIAGNOSTICS

Errors detected by the shell, such as syntax errors, cause the shell to return a non zero exit status. If the shell is being used non interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

BUGS

If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document. A garbage file /tmp/sh* is created, and the shell complains about not being able to find the file by another name.

NAME

shelltool – Run a shell (or other program) in the SunWindows environment

SYNOPSIS

shelltool [**–C**] [**program** [**args**]]

OPTIONS

–C Redirect system console output to this instance of the shelltool.

shelltool also takes generic tool arguments; see *suntools*(1) for a list of these arguments.

If a *program* argument is present, *shelltool* runs it. If there are no arguments, *shelltool* runs the program corresponding to your SHELL environment variable. If this environment variable is not available, then *shelltool* runs */bin/sh.*

DESCRIPTION

shelltool is a standard tool provided with the *SunWindows* environment.

When invoked, *shelltool* starts up an interactive character-terminal-based program, usually a shell, inside of a terminal emulator subwindow. If this program is a shell, it accepts commands and runs programs in the usual way.

shelltool's window may be positioned and manipulated in the same way as any tool window in the SunWindows environment. *shelltool* supports selections as described in *suntools*(1).

To invoke Tool Manager Functions, position the cursor on the name stripe or on the outside borders of *shelltool*'s window. The Tool Manager is not accessible from *shelltool*'s terminal subwindow because keystrokes are passed to the shell, or to the programs run from it. See *suntools*(1), *Terminal Emulators* for more information.

To run graphics programs, use *gfxtool* — see *gfxtool*(1).

Unless you instruct the system otherwise, it displays messages for the console by writing over the windows on the display. It bypasses all of the support for multiple windows sharing the bitmap. Solve this problem by starting a copy of *shelltool* with the -C argument; [that] redirects system console messages into that instance of the shelltool.

See *suntools*(1) for information on terminal emulators, and for an example showing how *shelltool* can be started automatically when the windowing environment starts up.

SEE ALSO

suntools(1)
gfxtool(1)

FILES

~/.ttyswrc
/usr/bin/shelltool
/usr/bin/suntools
/usr/demo/*
/usr/src/sun/suntool/shelltool.c

BUGS

This release has the following notable bugs:

(1) Remote login to another machine should be done with a terminal emulator subwindow matching the standard terminal size for the remote machine (for example, 34 by 80 characters for a Sun workstation). The remote machine does not understand terminals with

non-standard size.

(2) If more than 256 characters are input to a terminal emulator subwindow without an intervening newline, the terminal emulator may hang. If this occurs, display the Tool Manager Menu; the "TTY Hung?" submenu there has one item, "Flush input", that you can invoke to correct the problem.

NAME

size − size of an object file

SYNOPSIS

size [object **...**]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each object-file argument. If no file is specified, *a.out* is used.

SEE ALSO

a.out(5)

NAME

sleep — suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

setitimer(2), sleep(3)

BUGS

Time must be less than 2,147,483,647 seconds.

NAME

soelim — eliminate .so's from nroff input

SYNOPSIS

soelim [file ...]

DESCRIPTION

Soelim reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

```
.so somefile
```

when they appear at the beginning of input lines. This is useful since programs such as *tbl* do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (—) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using '´' instead of '.', that is,

```
´so /usr/lib/tmac.s
```

EXAMPLE

A sample usage of *soelim* would be

```
soelim exum?.n ¦ tbl ¦ nroff −ms ¦ col ¦ lpr
```

SEE ALSO

colcrt(1), more(1)

BUGS

The format of the source commands must involve no strangeness — exactly one space must precede and no spaces follow the file name.

NAME

sort – sort or merge files

SYNOPSIS

sort [**–mubdfinrt***x*] [**+***pos1* [**–***pos2*]] **...** [**–o** name] [**–T** directory] [file] **...**

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name '–' means the standard input. If no input *file*'s are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence.

The notation **+***pos1* **–***pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m***.***n*, optionally followed by one or more of the flags **bdfinr,** where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2*. A missing **.***n* means .0; a missing **–***pos2* means the end of the line. Under the **–t***x* option, fields are strings separated by *x*; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

OPTIONS

The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d 'Dictionary' order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b.**

r Reverse the sense of comparisons.

t*x* 'Tab character' separating fields is *x*.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

o *name*
name is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

T *directory*
directory argument is the name of a directory in which temporary files should be made.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort −u +0f +0 list
```

Print the password file (*passwd*(5)) sorted by user id number (the 3rd colon-separated field).

```
sort −t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort −um +0 −1 dates
```

FILES

/usr/tmp/stm*, /tmp/* first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **−c**.

BUGS

Very long lines are silently truncated.

NAME

sortbib – sort bibliographic database

SYNOPSIS

sortbib [–sKEYS] database ...

DESCRIPTION

Sortbib sorts files of records containing *refer* key-letters by user-specified keys. Records may be separated by blank lines, or by .[and .] delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so *sortbib* may not be used in a pipeline to read standard input.

By default, *sortbib* alphabetizes by the first %A and the %D fields, which contain the senior author and date. The –s option is used to specify new KEYS. For instance, –sATD will sort by author, title, and date, while –sA+D will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

Sortbib sorts on the last word on the %A line, which is assumed to be the author's last name. A word in the final position, such as "jr." or "ed.", will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the *nroff* convention "\0" in place of a blank. A %Q field is considered to be the same as %A, except sorting begins with the first, not the last, word. *Sortbib* sorts on the last word of the %D line, usually the year. It also ignores leading articles (like "A" or "The") when sorting by titles in the %T or %J fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, *sortbib* places that record before other records containing that field.

SEE ALSO

refer(1), addbib(1), roffbib(1), indxbib(1), lookbib(1)

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell, spellin, spellout — find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−d** hlist] [**−s** hstop] [**−h** spellhist] [file] ...

spellin [list]

spellout [**−d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (for example, thier=thy−y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell*. Both expect a list of words, one per line, from the standard input. *Spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option **−d**) the hash list.

OPTIONS

−v Print all words not literally in the spelling list, as well as plausible derivations from spelling list words.

−b Check British spelling. Besides preferring *centre*, *colour*, *speciality*, *travelled*, and so on, the **−b** option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

−x print every plausible stem with '=' for each word.

FILES

/usr/dict/hlist[ab]	hashed spelling lists, American & British
/usr/dict/hstop	hashed stop list
/usr/dict/spellhist	history file
/usr/dict/words	list of words
/usr/lib/spell	

SEE ALSO

deroff(1), sort(1), tee(1), sed(1)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.

British spelling was done by an American.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [**−a**] [**−k**] [**−n**] [**−p**] [**−x**]

DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

OPTIONS

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

−k The constant k used in the boundary value computation

$$y''_0 = ky''_1, \quad y''_n = ky''_{n-1}$$

is set by the next argument. By default $k = 0$.

−n Space output points so that approximately n intervals occur between the lower and upper x limits. (Default $n = 100$.)

−p Make output periodic, that is, match derivatives at ends. First and last input values should normally agree.

−x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1G)

DIAGNOSTICS

When data is not strictly monotonic in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

NAME

split − split a file into pieces

SYNOPSIS

split [−*number*] [*infile* [*outfile*]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000) onto a set of output files (as many files as necessary). The name of the first output file is *outfile*¦ with **aa** appended, the second file is *outfile***ab**, and so on lexicographically.

If no *outfile* is given, *x* is used as default (output files will be called ***x*aa**, ***x*ab**, etc.).

If no *infile* is given, or if − is given in its stead, then the standard input file is used.

OPTIONS

−*number* Number of lines in each piece.

NAME

strings – find printable strings in an object, or other binary, file

SYNOPSIS

strings [–] [**–o**] [*–number*] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null.

Strings is useful for identifying random object files and many other things.

OPTIONS

– Look everywhere in the file for strings. If this flag is omitted, *strings* only looks in the initialized data space of object files.

–o Precede each string by its **o**ffset in the file (in octal).

–number
Use *number* as the minimum string length rather than 4.

SEE ALSO

od(1)

BUGS

The algorithm for identifying strings is extremely primitive.

NAME

strip – remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **—s** option of *ld*.

FILES

/tmp/stm?	temporary file

SEE ALSO

ld(1)

NAME

stty – set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, and directs its output to the diagnostic output. With no argument, it reports the speed of the terminal and the settings of options which are different from their defaults. With the argument "all", all normally-used option settings are reported. With the argument "everything", everything *stty* knows about is printed.

OPTIONS

Options to *stty* are selected from the following set:

even allow even parity input
–even disallow even parity input
odd allow odd parity input
–odd disallow odd parity input
raw raw mode input (**no** input processing (erase, kill, interrupt, ...); parity bit passed back)
–raw negate raw mode
cooked same as '–raw'
cbreak make each character available to *read*(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed
–cbreak make characters available to *read* only when newline is received
–nl allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl accept only new-line to end lines
echo echo back every character typed
–echo do not echo characters
lcase map upper case to lower case
–lcase do not map case
tandem enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
–tandem disable flow control
–tabs replace tabs by spaces when printing
tabs preserve tabs
ek set erase and kill characters to ^H (control-H) and ^U

For the following commands which take a character argument *c*, you may also specify *c* as the "u" or "undef", to set the value to be undefined. A value of "^x", a 2 character sequence, is also interpreted as a control character, with "^?" representing delete.

erase *c* set erase character to *c* (default '^?').
kill *c* set kill character to *c* (default '^U').
intr *c* set interrupt character to *c* (default '^C').
quit *c* set quit character to *c* (default '^\').
start *c* set start character to *c* (default '^Q').
stop *c* set stop character to *c* (default '^S').
eof *c* set end of file character to *c* (default '^D').
brk *c* set break character to *c* (default undefined.) This character is an extra wakeup causing character.

cr0 cr1 cr2 cr3
select style of delay for carriage return (see *ioctl*(2))

nl0 nl1 nl2 nl3
select style of delay for linefeed

tab0 tab1 tab2 tab3

select style of delay for tab

ff0 ff1 select style of delay for form feed

bs0 bs1 select style of delay for backspace

tty33 set all modes suitable for the Teletype Corporation Model 33 terminal.

tty37 set all modes suitable for the Teletype Corporation Model 37 terminal.

vt05 set all modes suitable for Digital Equipment Corp. VT05 terminal

dec set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and "newcrt".)

tn300 set all modes suitable for a General Electric TermiNet 300

ti700 set all modes suitable for Texas Instruments 700 series terminal

tek set all modes suitable for Tektronix 4014 terminal

0 hang up phone line immediately

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 exta extb
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

The driver which supports the job control processing of *csh*(1) is fully described in *tty*(4). The options in the list below can only be selected by using the **new** option to *stty*(1).

new Use new driver (switching flushes typeahead).

crt Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)

crtbs Echo backspaces on erase characters.

prterase For printing terminal echo erased characters backwards within "\" and "/".

crterase Wipe out erased characters with "backspace-space-backspace."

−crterase
Leave erased characters visible; just backspace.

crtkill Wipe out input on like kill ala **crterase.**

−crtkill Just echo line kill character and a newline on line kill.

ctlecho Echo control characters as "^*x*" (and delete as "^?".) Print two backspaces following the EOT character (default '^D').

−ctlecho Control characters echo as themselves; in cooked mode EOT (default '^D') is not echoed.

decctlq After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.

−decctlq After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)

tostop Background jobs stop if they attempt terminal output.

−tostop Output from background jobs to the terminal is allowed.

tilde Convert "~" to "`" on output (for Hazeltine terminals).

−tilde Leave poor "~" alone.

flusho Output is being discarded usually because user hit '^O' (internal state bit).

−flusho Output is not being discarded.

pendin Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).

−pendin Input is not pending.

intrup Send a signal (SIGTINT) to the terminal control process group whenever an input record (line in cooked mode, character in cbreak or raw mode) is available for reading.

−intrup Don't send input available interrupts.

mdmbuf Start/stop output on carrier transitions (not implemented).

−mdmbuf
Return error if write attempted after carrier drops.

litout Send output characters without any processing.

—litout Do normal output processing, inserting delays, etc.
nohang Don't send hangup signal if carrier drops.
—nohang Send hangup signal to control process group when carrier drops.
etxack Diablo style etx/ack handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed.

susp *c* set suspend process character to *c* (default '^Z').
dsusp *c* set delayed suspend process character to *c* (default '^Y').
rprnt *c* set reprint line character to *c* (default '^R').
flush *c* set flush output character to *c* (default '^O').
werase *c* set word erase character to *c* (default '^W').
lnext *c* set literal next character to *c* (default '^V').

SEE ALSO

ioctl(2), tset(1), tty(4)

NAME

su – substitute user id temporarily

SYNOPSIS

su [*username*] [–] [**–c** *command*] [**–f**]

DESCRIPTION

Su changes your login to that of the specified *username*. *Su* asks for the password, just as if you were logging in as *username*, and, if the password is given, changes to that *username* and invokes the shell specified in the password file for that *username*, without changing the current directory. The user environment is thus unchanged except for **HOME** and **SHELL**, which are taken from the password file for the user being substituted (see *environ*(5)). The new user ID stays in force until the shell exits.

If no *username* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

OPTIONS

- The **-** flag performs a complete login. That is, it moves to the home directory, reads the *.login* file, and reads the *shell.*

–c *command*
execute *command* after logging in as the new user.

–f Perform a fast login. That is, do not change directories, do not read the *.cshrc* file, and do not read the .login file for the new user ID to configure the new shell.

If the – and **-f** flags are omitted, only the *shell.*

SEE ALSO

sh(1), csh(1)

NAME

sum − sum and count blocks in a file

SYNOPSIS

sum file

DESCRIPTION

Sum calculates and displays a 16-bit checksum for the named file, and also displays the size of the file in kilobytes. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinguishable from end of file on most devices; check the block count.

NAME

sun – is current machine a Sun Workstation

SYNOPSIS

if sun; then ...; fi *(sh)*

if { sun } then *(csh)*
...
endif

DESCRIPTION

The *sun* command is, on Sun Workstations, the same as *true*(1); on VAX'es and other machines it is the same as *false*(1).

SEE ALSO

false(1), true(1), vax(1)

NAME

suntools − Run the SunWindows environment

SYNOPSIS

suntools [**−n** ¦ **−s** *startup file*] [**−d** *display device*] [**−m** *mouse device*]
[**−k** *keyboard device*] [**−b** *red green blue*] [**−f** *red green blue*] [**−i**] [**−p**] [**−** [**B** ¦ **F** ¦ **P**]]

GETTING STARTED

suntools starts up the SunWindows environment and awaits your directions. You will often give directions by selecting a menu item. If your */dev* directory has not been initialized for the window environment, *suntools* will fail; see *Initialization* below for details. If there is a file called *.suntools* in your home directory, *suntools* starts the application programs (*tools*) specified in that file. You can find the format of the *.suntools* file described in detail under *Start-up Processing* below.

The *SunWindows* system is ready when:

1) the screen is painted with a standard pattern (usually light gray, see OPTIONS)

2) an arrow-shaped cursor appears on the screen, tracking motions of the mouse. An optical mouse must be on its special pad before it can be tracked. Sometimes the cursor will not move at start-up; moving the mouse in large circles for a few seconds brings the cursor to life.

See *Multiple/Color Displays* for information about running *suntools* on more than one display at the same time or on color displays.

OPTIONS

−n Bypasses startup processing by ignoring the *.suntools* file. See *Startup Processing* for more on startup processing.

−s *startup file*
Read startup commands from *startup file* (instead of *~/.suntools*).

−d *display device*
Use *display device* as the output device on which to run. If this option is not specified, SunWindows output goes to the default frame buffer device, */dev/fb.*

−m *mouse device*
Use *mouse device* as the system pointing device (locator). If this option is not specified, SunWindows uses the default mouse device, */dev/mouse.*

−k *keyboard device*
Accept keyboard input from *keyboard device.* If this option is not specified, SunWindows uses the default keyboard device, */dev/kbd.*

−b *red green blue*
Specifies the values of the *red, green* and *blue* components of the background color. If this option is not specified, each component of the background color is 255 (white). See the **−f** option for more details.

−f *red green blue*
Specifies the values of the *red, green* and *blue* components of the foreground color. For most color screens, values must be between 0 and 255 where 0 is the absence of color and 255 is the maximum intensity of that color. If this option is not specified, each component of the foreground color is 0 (black).

−i Invert the background and foreground colors used on the screen. On a monochrome monitor, this option provides a video reversed image. On a color monitor, colors that are not used as the background and foreground are not affected.

−p Displays the name of the window device used for the *suntools* window.

−B Uses the background color for the *suntools* window color.

−F Uses the foreground color for the *suntools* window color.

−P Uses a stipple pattern for the *suntools* window color. If no **-F**, **-B** or **-P** option is given, **-P** is assumed.

DESCRIPTION

Windows

The *SunWindows* environment always has one window open which covers the whole screen; this is called the *Root Window*. A solid color is the only content of the Root Window; it serves as a *backdrop* for the *SunWindows* environment. Tools that run in the environment are given their own windows which lie on top of portions of the Root Window (and possibly other tools). A window obscures any portions of another window which lies below it. Only the topmost window is guaranteed to be fully visible, and the Root Window color shows through only in areas which have no tool windows on them.

Input to Windows

When you type on the keyboard, move the mouse, or press mouse buttons, be sure your mouse cursor is inside the window you want to affect. Otherwise, your input gets lost or sent to a different window.

Your input actions are queued for each window, with mouse motions and button pushes integrated with keystrokes. This means that you can type some characters to one window, move the mouse to another window, then type or use the mouse in the second window before the first window has processed its input. Each tool's input handling is described with the rest of its behavior in its section within the *Commands Reference Manual.*

The Mouse

Use the mouse buttons as follows:

Left button	(the *select* button)	Click once to select or choose objects.
Middle button	(the *adjust* button)	Click once to shorten or lengthen your selection.
Right button	(the *menu* button)	Depress and hold down to invoke menus.

The exact interpretation of your actions depends on the type of window receiving them. Any window type or tool that interprets mouse input differently has the differences documented in its separate section.

Menus

You can invoke many functions in *suntools* by using the mouse and a *pop-up menu*. When you press down on the *menu* button and hold it down, the menu(s) appropriate to the window appear. Remember that the mouse cursor must be in the window, or you may get unexpected results.

As you press the menu button, the menu appears on the screen near the current location of the mouse. The cursor changes to a right-pointing arrow called the *pointer.* The menu remains on the screen as long as you hold the menu button down. Move the pointer with the mouse. To invoke a menu item, point at it (it will be highlighted), then release the menu button.

You may see more than one menu presented simultaneously; generally this occurs when several different classes of actions are possible in the current context. The menus are presented in a stack, with the label of each menu visible, and with the current menu on top so its items are also visible. To bring another menu to the top, thus making its items available, invoke its label as you would a menu item. That is, position the cursor in the label of the desired menu and release the menu button. Then push the menu button again. The stack of menus is repainted with the selected menu on top, and menu operations may continue. Another way to bring a rear menu forward is to press and release the *select* mouse button while the pointer is over the label of the desired menu, holding the *menu* button down the whole time. This rearranges the menus without removing them from the screen.

The Suntools Menu

The Root Window's default menu (the Suntools Menu) contains four job-control functions. To invoke it, move the pointer anywhere in the Root Window, then depress and hold down the menu button.

The items in the Suntools Menu are:

New Shell
Creates a new Shell Tool.
A tool running a new copy of the shell appears on the screen. The window is placed on the screen on top of everything it overlaps. Its initial size can display 80 columns and 34 rows of characters.

New Graphics
Creates a new Graphics Tool.
Creates a new tool window appropriate for running graphics programs. It is placed on the screen on top of everything it overlaps.

Exit
Exits the *suntools* program.
Closes all tool windows and kills their associated processes. You return to the shell which invoked *suntools*. This command requires confirmation: When it prompts you, press the left mouse button to complete the Exit command; press the right button to cancel.

ReDisplay All
Redraws all the contents of the screen.
Use this to repair damage done by processes that wrote to the screen without consulting the SunWindows system. See *shelltool*(1) for details on how to avoid this.

The Root Window ignores most keyboard input; if you type while the cursor is over an exposed portion of the Root Window, the keystrokes are simply discarded (without any audible or visible feedback).

The Tool Manager

Tools interpret most of your input in their own fashion; for instance, many tools incorporate a terminal emulator running a shell. Tools also provide you with a small set of universal functions through the Tool Manager menu.

You can invoke the Tool Manager menu when the cursor is over a portion of the tool which does not impose some other interpretation on your input. For any tool, the tool namestripe (black stripe holding the tool's name), the border stripes of the window, and the whole of the tool's icon are such areas.

The items in this menu are:

Close (or Open)
Only one of "Close" or "Open" appears in the menu, as appropriate to the current state of the window. Close shrinks the tool to a small image on the screen. Open reopens a closed window and places it in the spot it occupied when

it was last open.
when the window is closed, the tool's process(es) continue to run. Closed windows are placed by default in the lower left of the screen, filling in rows to the right, and then bottom-to-top, in the order they are created. You can move a closed window just like an open window. If you reopen it and close it again, it returns to its last position on the screen, without affecting its size or contents.

Move — Move the tool window to another spot on the screen.
When invoked, Move instructs you to grasp the window by depressing and holding down either the left or middle mouse buttons, or to cancel the operation by pressing the right button. Depressing any mouse button removes these instructions from the screen.

When you depress either the left or middle button, a box called a *bounding box* outlines the window. The bounding box tracks the mouse as long as you hold that button down. When you press the button, the cursor location determines how the window is moved. If the mouse cursor is near a corner when you press the button, that corner of the box attaches to the mouse cursor. If it is in the middle third of a side, then the midpoint of the box's side attaches to the mouse cursor.

Position the bounding box where you want the window to go and release the mouse button. If the window is on top of or underneath other windows before you move it, it will be on top of or underneath the same windows after you move it.

Stretch — Shrinks or stretches the size of a window on the screen.
Stretch works almost like Move. Depress and hold down either the left or middle mouse buttons to grasp the window. Cancel the operation by pressing the right button.

When you depress either the left or middle button, a bounding box outlines the window. The bounding box tracks the mouse as long as you hold that button down. When you press the button, your action determines how the window is stretched. If the mouse cursor is near a corner when you press the button, both of the sides that form that corner are adjusted; the opposite corner remains fixed. If it is in the middle third of a side, then only that side is adjusted.

Adjust the bounding box to the shape you want the window to take, and release the mouse button. If the window is on top of or underneath other windows before you move it, it will be on top of or underneath the same windows after you stretch it.

Expose — Brings the window to 'the top of the heap'.
The whole window becomes visible, and occludes any window it happens to overlap on the screen. Its position on the screen does not change.

Hide — Puts the window on the 'bottom of the heap'.
The window is occluded by any window which overlaps it. Its position on the screen does not change.

ReDisplay — Redraws the contents of the window.

Quit — Notifies the tool to terminate gracefully.
This command requires the same type of confirmation as the Exit command in the Suntools Menu.

In many multi-subwindow tools, you can adjust the boundary between two subwindows up or down without changing the overall size of the tool. Depress the middle mouse button over the boundary. A bounding box is drawn for the subwindow above or to the left of the selected boundary. Now adjust the size of that subwindow, exactly as with the Stretch operation. When the button is released, that subwindow is adjusted to the size indicated, and the remaining area of the tool is allotted to the other subwindows.

Tool Management Accelerators

Accelerators are provided for some of the Tool Manager functions. You can invoke these functions quickly with a simple button push in the tool window, rather than displaying a menu.

The accelerators for the various functions are:

Open — To open an iconic tool, click the select mouse button when the cursor is over the icon.

Move — To invoke Move, depress the middle mouse button while the cursor is in the tool window's name stripe or outer boundary. A bounding box displays and tracks the mouse as long as you hold the middle button down. Release the button when the box indicates the spot where you want to place the window.

Expose — To expose a window, click the left mouse button at an open tool window while the cursor is on the tool's name stripe or outer boundary.

Help — To get help, type a question mark to a tool window. A help message explaining the material covered here appears.

Generic Tool Arguments

Most tools now take the following arguments in their command lines:

FLAG	(LONG FLAG)	ARGS	NOTES
-Ww	(-width)	columns	
-Wh	(-height)	lines	
-Ws	(-size)	x y	x and y are in pixels
-Wp	(-position)	x y	x and y are in pixels
-WP	(-icon_position)	x y	x and y are in pixels
-Wl	(-label)	"string"	
-Wi	(-iconic)		makes the tool start iconic
-Wt	(-font)	filename	
-Wf	(-foreground_color)	red green blue	0-255 (no color-full color)
-Wb	(-background_color)	red green blue	0-255 (no color-full color)
-Wg	(-set_default_color)		(apply color to subwindows too)
-WI	(-icon_image)	filename	(for tools with non-default icons)
-WL	(-icon_label)	"string"	(for tools with non-default icons)
-WT	(-icon_font)	filename	(for tools with non-default icons)
-WH	(-help)		print this table

Each flag option may be specified in either its short form or its long form; the two are completely synonymous.

Terminal Emulators

Several tools provide a terminal emulator running a shell in some part of their window. When you type to this *subwindow*, the keystrokes pass through to the shell program running in it, rather than being handled by the tool. These tools provide Tool Manager functions only when the cursor is in their name stripe or borders. Similarly, output from the shell program displays in the subwindow.

Whenever a tty subwindow is created, the startup file ~/.ttyswrc is parsed for tty subwindow-specific parameters. A sample .ttyswrc file may be found at the end of this section. The command format of this file is:

#	Comment.
set *variable*	Turn on the specified variable.
mapi *key text*	When *key* is typed pretend *text* was typed.
mapo *key text*	When *key* is typed pretend *text* was output.

The only currently defined "variable" is "pagemode". "Key" is one of L1-L15, F1-F15, T1-T15, R1-R15, LEFT, or RIGHT (see note below). "Text" may contain escapes such as \E, \n, ^X, etc. (escape, newline, control-X, respectively). See *termcap*(5) for the format of the string escapes that are recognized. Note that "mapi" and "mapo" may be replaced by another keymapping mechanism in the future.

NOTE: When using the default kernel keyboard tables, the keys L1, LEFT, RIGHT, BREAK, R8, R10, R12, and R14 cannot be mapped in this way because they send special values to the terminal emulator subwindow. See *kbd*(5) for more information.

It is possible to have terminal-based programs drive the tool in which its tty subwindow resides by sending it special escape sequences. These escape sequences may also be sent using the "mapo" function described above. The following functions pertain to the tool in which the tty subwindow resides, not the tty subwindow itself.

\E[1t	– open
\E[2t	– close (become iconic)
\E[3t	– move, with interactive feedback
\E[3;TOP;LEFTt	– move, to TOP LEFT (pixel coordinates)
\E[4t	– stretch, with interactive feedback
\E[4;WIDTH;HTt	– stretch, to WIDTH HT size (in pixels)
\E[5t	– top (expose)
\E[6t	– bottom (hide)
\E[7t	– refresh
\E[8;ROWS;COLSt	– stretch, to ROWS COLS size (in characters)
\E[11t	– report if open or iconic by sending \E[1t or \E[2t
\E[13t	– report position by sending \E[3;TOP;LEFTt
\E[14t	– report size in pixels by sending \E[4;WIDTH;HTt
\E[18t	– report size in characters by sending \E[8;ROWS;COLSt
\E[20t	– report icon label by sending \E]Llabel\E\
\E[21t	– report tool header by sending \E]llabel\E\
\E]l<text>\E\	– set tool header to <text>
\E]I<file>\E\	– set icon to the icon contained in <file>; <file> must be in *icontool* output format
\E]L<label>\E\	– set icon label to <label>
\E[>OPT;...h	– turn OPT on (OPT = 1 => pagemode), e.g., \E[>1;3;4h
\E[>OPT;...k	– report OPT; sends \E[>OPTl or \E[>OPTh for each OPT
\E[>OPT;...l	– turn OPT off (OPT = 1 => pagemode), e.g., \E[>1;3;4l

As an example of using this facility, the following aliases can be put into your ~/.cshrc file:

```
# dynamically set the name stripe of the tool:
alias header 'echo -n "^[]l\!*^[\"'
# dynamically set the label on the icon:
alias iheader 'echo -n "^[]L\!*^[\"'
# dynamically set the image on the icon:
alias icon 'echo -n "^[]I\!*^[\"'
```

A sample .ttyswrc file follows:

```
#
#    ttysubwindow startup file
#
set      pagemode
# Top:
mapo     T1      \E[5t
# Close:
mapo     T2      \E[2t
# Move:
mapo     T3      \E[3t
# Stretch:
mapo     T4      \E[4t
# Bottom:
mapo     T5      \E[6t
# Refresh:
mapo     T6      \E[7t
# Move (non-interactive) to top left:
mapo     T7      \E[3;1;1t
# Stretch in chars (non-interactive) to half high:
mapo     T8      \E[8;10;80t
# Stretch in chars (non-interactive) to normal size:
mapo     T9      \E[8;34;80t
# Commands (very left keys, not setup):
mapi     L3      jobs
mapi     L5      mail
mapi     L7      ls -F
mapi     L9      more errs
# Editing:
# Bracket word in italic escapes while in vi:
mapi     R4      i\fIEea\fPE
# Bracket word in bold escapes while in vi:
mapi     R5      i\fBEea\fPE
# Mail:
mapi     R1      dt
mapi     R2      s +planning\ndt\n
mapi     R3      s +bugs\ndt\n.sp
```

Selections

Terminal subwindows support a facility called *selection,* which provides for limited inter-tool communication and mouse-oriented text manipulation. A *selection* is a span of characters which you can manipulate. To make a selection:

- Press the *select* (left) mouse button while the tip of the cursor is over the desired character. Your selection becomes highlighted. This feedback helps you see what you're doing. Any previous selections are de-selected; the highlighting around the old selection disappears. Move the mouse with the select button down, and the selection changes. Release the select button to complete the selection.
- Press the *adjust* (middle) mouse button down and move the mouse to change the span of characters that you select. Release the button. All characters, from the one you originally selected through the one indicated when you released *adjust,* are selected. The highlighting indicating the selection adjusts to reflect its new contents.

You can also adjust your selection by *multi-clicking.* For example, if you select a character and then click the left or middle mouse button over it within .5 seconds, the highlighting adjusts to select a word. Click again within .5 seconds, and the highlighting adjusts to select a line. Click yet again, and the highlighting now shows you selected a paragraph. Click one more time, and the selection shrinks back to the original character you selected.

You can only select characters which are displayed in a terminal subwindow. You can select characters obscured by another window if they lie between the characters you chose as the endpoints to the selection.

The highlighting indicating the selection vanishes if:

- you move the cursor out of the subwindow which holds the selection,
- you type any key, or
- any new output is written to the window which holds the selection.

However, even if the selection highlighting disappears, *the selection still exists.* You can use it until you explicitly make another selection. The fact that the selection highlighting disappears in these circumstances reflects an incomplete selection implementation.

To manipulate your selection, press the menu button over the terminal subwindow. A *ttysw* menu appears with the menu items discussed below:

Stuff Select Stuff, and the characters in the selection are copied to the window as though they had been typed at the keyboard. The window in which you invoke Stuff does not have to be the same as the one in which you made the selection. You can copy text between windows with this facility.

If you are an expert user, you may want to accelerate the Stuff process. If you hold down the shift key while you press the right mouse button, you can stuff a selection without selecting Stuff from the menu.

Page Mode On
Select Page Mode On to prevent voluminous tty subwindow output from scrolling off the screen. Page Mode can save you from redoing a command with a pipe to *more* in such cases.

When Page Mode is on, a tiny stop sign appears on the screen when a command generates a screenful of output. To restart output, type any key, or select the *Continue* menu item discussed below.

Page Mode Off
After you turn Page Mode on, you can select Page Mode Off to turn it off.

Continue
Select Continue to restart output halted by Page Mode.

You can use the selection mechanism for building up complex command sequences and to avoid repetitive keyboard input. Selection facilities will be expanded in future releases.

Other Aspects of Terminal Emulators

Currently, only *shelltool* and *gfxtool* contain terminal emulator subwindows. When you call one of these tools with command line arguments, any arguments not specific to the tool pass to the terminal emulator. The terminal emulator runs the program named by the first of these arguments. If you don't give any arguments, then the terminal emulator runs the program described by your SHELL environment value. If this environment value is not available, then */bin/sh* is run. Note: To run a C shell using its -c option in either of these tools, you must explicitly specify **csh -c** on the command line when you start up the tool.

Start-up Processing: The '.suntools' File

suntools can set up a predefined arrangement of windows for you when it starts up. It does this by reading the file *.suntools* in your home directory and following the instructions there. The **−s** flag on the command line indicates that *suntools* should read an alternative file. The **−n** switch suppresses this start-up processing altogether.

Each line in the file is a command line corresponding to a tool to be started up. These command lines are similar to those given to a shell, but no special character interpretation is performed. Use the generic tool arguments to place the tool on the screen (see *Generic Tool Arguments*). Comment lines are preceded by a #.

For example:

```
shelltool -C -Wi
shelltool
clocktool -Wi
```

starts up a *shelltool* as the console in its iconic form, a *shelltool* to be used for interactive command input and a clocktool, also in its iconic form. Note that the system assigns the positions of windows and icons in this case, since no positioning arguments were given.

As an alternative, a command line may be a string followed by 9 decimal numbers, as follows:

path nl nt nw nh il it iw ih I args

where *path* is the file name of the tool to be run, *nl*, *nt*, *nw*, and *nh* are decimal numbers which define the window for the normal (open) tool, as left edge, top edge, width, and height; *il*, *it*, *iw*, and *ih* are the same for the iconic form of the tool; and *I* is a boolean flag which should be non-zero if the tool is to be started in its *iconic* form. The remainder of the line is passed to the the program running in the window as command-line arguments. The origin (0,0) is the upper left-hand corner of the screen. If a size parameter is equal to −1 then the Root Manager assigns an appropriate value.

- You can include comments lines in a *.suntools* file by preceding them with a #.
- You can place command lines in a *.suntools* file, but no interpretation *ala* shell is done. You can combine this facility with the generic command line tool arguments to place tools.

Consider a *.suntools* file containing:

```
shelltool   0    72   -1    728  4     4    64   64   0   vi $HOME/sked
shelltool   372  0    -1    792  -1    -1   64   64   0   -C
clocktool   4    4    180   40   232   4    64   64   1
```

This starts *suntools* with three tools active:

- A *shelltool* in a window 38 by 80 characters (in the default font). It comes up in the lower left corner of the screen, which leaves enough space above it to fit some icons. The *shelltool* is running the editor *vi* on a file named 'sked' in your home directory. The string 'vi $HOME/sked' passes as a command to be executed by the shell running in the tool. When this shell becomes iconic, it moves to the upper left corner of the screen.
- Another *shelltool* extends nearly the full height of the screen (48 lines), and butts against the right edge. The system places its icon when this shell becomes iconic. The shell is ready for your input as soon as it appears on the screen. Note, however, that all your messages are directed to this console shell. If you hate being interrupted by biff, you should either add another shell to your *.suntools* file, or plan to start another shell in which to work as soon as Suntools comes up on the screen.
- A *clocktool* starts out iconic, about a quarter of the way across the top of the screen.

The program *toolplaces* in the directory */usr/bin* can help you construct the *.suntools* file; it writes a description of the windows that exist at its run time on the standard output.

Customizing the Root Manager Menu

A file in /usr/lib called "rootmenu" contains a description of the menu displayed over the gray root window. Setting the ROOTMENU parameter in your environment to another file changes the place from which the menu description is gleaned. You can customize the menu by modifying the file. Lines in the file start with the string to display in the menu followed by a command (strings with embedded blanks are delimited with double quotes). Here are the commands that you can supply. Comment lines start with a #:

EXIT Exit the suntools program, after user confirmation. To add this command, place the line

```
Exit    EXIT
```

in the file.

REFRESH

Redraw the entire screen. To add this command, place the line

```
ReDisplay REFRESH
```

in the file.

MENU Stack a menu on the pile with the Suntools menu. Get the menu contents from the filename that follows. Name the menu with the string preceding. To add this command, place the line

```
"More Tools"      MENU /usr/tmp/rootmenutool
```

in the file.

Lines without recognized commands are treated as command lines and executed:

```
Shell  /usr/bin/shelltool
Lock    /usr/bin/lockscreen -r
```

No interpretation *ala* shell is done.

Changing the System Font

The system has a default font built into it. To use another font, set the environment variable DEFAULT_FONT to the name of the file containing the desired font. You have to set DEFAULT_FONT in the shell from which you invoke the program. You can find a small number of valid alternative fonts in the directory */usr/bin/fonts/fixedwidthfonts.*

Multiple/Color Displays

The *suntools* program runs on either a monochrome or color screen. Each screen on a machine may have its own invocation of *suntools* running on it. The keyboard and mouse input devices are shared among multiple screens. The mouse cursor slides from one screen to another when you move the cursor off the edge of a screen.

A common multiple display configuration is one monochrome and one color screen. You could set up an instance of *suntools* on each screen in the following way:

1) Invoke suntools on the monochrome display by running "suntools". This starts *suntools* on the default frame buffer named */dev/fb.*

2) In a Shelltool, run "suntools -d /dev/cgone0". This starts *suntools* on a Sun—1 color screen named /dev/cgone0.

3) In a Shelltool on the monochrome screen, run "adjacentscreens /dev/fb -r /dev/cgone0". This sets up cursor tracking so that the cursor slides from the monochrome screen to the color screen when you move the cursor off the right hand side of the monochrome screen. Similarly, the cursor slides from the color screen to the monochrome screen when you move the cursor off the left hand side of the color screen.

Getting Out

To exit any tool, invoke the Quit command in the Tool Manager Menu as described above. Typing ^D to the shell in a terminal subwindow also makes the tool go away. To exit the entire window system, invoke Exit in the Suntools menu as described above. Make sure that all windows are in a safe condition (for example, editors have written out all changes) first.

You can exit *suntools* via the keyboard by typing ^D followed by ^Q. There is no confirmation. This facility provides an escape hatch if you inadvertently start *suntools* when no mouse is attached to the system.

Initialization

Before you can run *suntools*, you have to create the appropriate devices for it in the */dev* directory. You only need to do this once on each machine. To create the devices,

- set your user id to *root*,
- change directory to */dev*, and
- execute the shell script 'MAKEDEV win0'.

If you tend to use more than 32 windows, run the script 'MAKEDEV win1' as well.

Suntools also needs the file */etc/utmp* to have *read* and *write* permission for all users. It should have been installed with these permissions, but if not, you need to use *chmod* to change the permissions.

SEE ALSO

Utility programs that run in the SunWindows environment:

adjacentscreens(1)
clocktool(1)
coretool(1)
dbxtool(1)
fonttool(1)
gfxtool(1)
icontool(1)
lockscreen(1)
perfmeter(1)
perfmon(1)
shelltool(1)
tektool(1)

FILES

~/.suntools
~/.ttyswrc
/usr/bin/suntools
/usr/bin/toolplaces
/usr/lib/fonts/fixedwidthfonts/*
/usr/lib/rootmenu
/usr/src/sun/suntool/*
/dev/win*x*
/dev/ptyp*x*
/dev/ttyp*x*
/dev/fb
/dev/kbd

/dev/mouse
/dev/MAKEDEV
/etc/utmp

BUGS

This release has the following notable bugs:

(1) Messages from the kernel ignore window boundaries unless console messages have been redirected, thus trashing the display. Recover from this by invoking the 'ReDisplay All' item on the Root Manager menu. Then start a *shelltool* with the -C option to redirect the console output.

(2) Acceptable performance on realistic tasks (such as editing with *vi* while *make* performs a C compilation) requires about 600K of available user memory. In some cases, the kernel may have to be reconfigured to make that space available. See the *System Manager's Guide*.

NAME

sync — update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to ensure all disk writes have been completed before the processor is halted in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8), cron(8)

NAME

syslog – make system log entry

SYNOPSIS

syslog [**–p**] [**–i** *name* [[*–level*] [–] [*message* **...**]

DESCRIPTION

Syslog sends the specified message (or *stdin* if – is specified) as a system log entry to the syslog daemon. The log entry is sent to the daemon on the machine specified by the *loghost* entry in the */etc/hosts* file.

OPTIONS

–p *Syslog* will log its process id in addition to the other information.

–i *name*
The specified name will be used as the "ident" for the log entry.

–level The message will be logged at the specified level. The level can be specified numerically, in the range 1 through 9, or symbolically using the names specified in the include file /usr/include/syslog.h (with the leading LOG_ stripped off). "*syslog* -HELP" will list the valid symbolic level names. Only the superuser can make log entries at levels less than or equal to SALERT.

– Each line of the standard input is sent as a log entry.

FILES

/usr/etc/in.syslog	syslog daemon
/usr/include/syslog.h	for names of logging levels

SEE ALSO

syslog(3), syslog(8)

NAME

tail – display the last part of a file

SYNOPSIS

tail [±*number*[**lbc**][**fr**]] [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.

OPTIONS

Options are all jammed together, not specified separately with their own – signs.

+*number*
: Begin copying at distance +*number* from the beginning of the file. *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.

–*number*
: Begin copying at distance –*number* from the end of the file. *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.

r
: Copy lines from the end of the file in **r**everse order. The default for **r** is to print the entire file this way.

f
: **F**ollow the file as it grows, that is, don't quit at end of file, but rather wait and try to read repeatedly in hopes that the file will grow.

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

talk – talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

Talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
user@host

though *user@host* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, *talk* sends the message:

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing:

```
tutorial% talk your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as their login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control-L redraws the screen, while your erase, kill, and word kill characters will work in *talk* as normal. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg*(1) command. At the outset talking is allowed. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent messy output.

FILES

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

SEE ALSO

mesg(1), who(1), mail(1), write(1)

NAME

tar – tape archiver

SYNOPSIS

tar –txruc[ovwfblmhpB0-9i] [tarfile] [blocksize] file1 file2 ... **–C** dir file*n* ...

DESCRIPTION

Tar saves and restores multiple files on a single *tarfile* (usually a magnetic tape, but it can be any file). *Tar*'s actions are controlled by its first argument, the *key*, a string of characters containing exactly one function letter from the set **rxtuc** and one or more optional *function modifiers*. Other arguments to *tar* are file or directory names specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

FUNCTION LETTERS

r Write the named files on the end of the *tarfile*. Note that this option *does not work* with quarter-inch archive tapes.

x Extract the named files from the *tarfile*. If the 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 file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier versions.

t List the names of the specified files each time they occur on the *tarfile*. If no file argument is given, all of the names on the *tarfile* are listed.

u Add the named files to the *tarfile* if they are not there or have been modified since last put on the *tarfile*. Note that this option *does not work* with quarter-inch archive tapes.

c Create a new *tarfile* and write the named files onto it.

FUNCTION MODIFIERS

0,...,9
Select an alternate drive on which the tape is mounted. The default is drive 0 at 1600 bpi, which is normally */dev/rmt8*.

f Use the next argument as the name of the archive instead of */dev/rmt8*. If the name of the file is '–', *tar* writes to standard output or reads from standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain. *Tar* can also be used to copy hierarchies with the command:

```
tutorial% cd fromdir; tar cf – . ¦ (cd todir; tar xfBp –)
```

o Suppress information specifying owner and modes of directories which *tar* normally places in the archive. Such information makes former versions of *tar* generate an error message like:

```
'<name>/: cannot create'
```

when they encounter it.

v Normally *tar* does its work silently; the **v** (verbose) option displays the name of each file *tar* treats, preceded by the function letter. When used with the **t** function, **v** displays the *tarfile* entries in a form similar to **ls –l**.

w Wait for user confirmation before taking the specified action. If you use **w**, *tar* displays the action to be taken followed by the file name, and then waits for a 'y' response to proceed. No action is taken on the named file if you type anything other than a line beginning with 'y'.

b Use the next argument as the blocking factor for tape records. The default blocking factor is 20 blocks. The block size is determined automatically when reading tapes (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to the program at the time it runs. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

l Display error messages if all links to dumped files cannot be resolved. If **l** is not used, no error messages are printed.

m Do not restore modification times of extracted files. The modification time will be the time of extraction.

h Follow symbolic links as if they were normal files or directories. Normally, *tar* does not follow symbolic links.

p Restore the named files to their original modes, ignoring the present umask(2). Setuid and sticky information are also restored if you are the super-user. This option is only useful with the **x** key letter.

B Force *tar* to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that *tar* can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming.

I Ignore directory checksum errors.

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:

```
tutorial% tar c −C /usr include −C /etc .
```

If you get a table of contents from the resulting *tarfile*, you will see something like:

```
include/
include/a.out.h
and all the other files in /usr/include
./
./chown
and all the other files in /etc
```

Note that the **−C** option only applies to *one* following directory name and *one* following file name.

EXAMPLES

Here is a simple example using *tar* to create an archive of your home directory onto */dev/rmt0*:

```
tutorial% cd                        position yourself in your home directory
tutorial% tar cvf /dev/rmt0 .       create the archive
          lots of messages from tar
tutorial%
```

The **c** option means create the archive; the **v** option makes *tar* tell you what it's doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (*/dev/rmt0* in this example).

Now you can read the table of contents from the archive like this:

```
tutorial% tar tvf /dev/rmt0         display table of contents of the archive
          lots of messages from tar
tutorial%
```

Where the **t** option is for displaying the table-of-contents of the archive. You can extract files from the archive like this:

```
tutorial% tar xvf /dev/rmt0         extract files from the archive
          lots of messages from tar
tutorial%
```

Where the **x** option is for extracting files from the archive.

If there are multiple archive files on a tape, each is separated from the following one by an end-of-file marker. *Tar* does not read the end-of-file mark on the tape after it finishes reading an archive file because *tar* looks for a special header to decide when it has reached the end of the archive. Now if you try to use *tar* to read the next archive file from the tape, *tar* doesn't know enough to skip over the end-of-file mark and tries to read the end-of-file mark as an archive instead. The result of this is an error message from *tar* to the effect:

```
          tar: blocksize=0
```

This means that to read another archive from the tape, the user must skip over the end-of-file marker before starting another *tar* command. You can achieve this via the *mt* command, as shown in the example below. Assume that you are reading from */dev/nrmt0.*

```
tutorial%   tar xvfp /dev/nrmt0       read first archive from tape
            lots of messages from tar
tutorial%   mt fsf 1  skip over the end-of-file marker
tutorial%   tar xvfp /dev/nrmt0 read second archive from tape
            lots of messages from tar
tutorial%
```

Finally, here is an example using *tar* to transfer files across the Ethernet. First, here is how to dump files from the local machine (tutorial) to a tape on a remote system (krypton):

```
tutorial%   tar cvfb  –  20  files ¦ rsh  krypton  dd of=/dev/rmt0  obs=20b
            lots of messages from tar
tutorial%
```

In the example above, we are *creating* a *tarfile* with the **c** key letter, asking for *verbose output from tar* with the **v** option, specifying the name of the output *tarfile* via the **f** option (the standard output is where the *tarfile* appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the *tar* command *and* on the *dd* command.

Now, here is how to use *tar* to get files from a tape on the remote system (krypton) back to the local system (tutorial):

```
tutorial%   rsh  krypton  dd  if=/dev/rmt0  bs=20b ¦ tar xvBfb  – 20  files
            lots of messages from tar
tutorial%
```

In the example above, we are *extracting* from the *tarfile* with the **x** key letter, asking for *verbose output from tar* with the **v** option, specifying the name of the input *tarfile* via the **f** option (the standard input is where the *tarfile* appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option.

FILES

/dev/rmt? half-inch magnetic tape interface
/dev/rar? quarter-inch magnetic tape interface
/dev/rst? SCSI tape interface
/tmp/tar*

SEE ALSO

tar(5), cpio(1), dump(8), restore(8)

DIAGNOSTICS

Complains about bad key characters and tape read/write errors.
Complains if enough memory is not available to hold the link tables.

BUGS

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

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.

There is no way to selectively follow symbolic links.

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [**−ms**] [**−mm**] [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input *files* are copied to the standard output, except that lines between .TS and .TE command lines are assumed to describe tables and are reformatted. Details are given in the *tbl*(1) reference manual.

If no arguments are given, *tbl* reads the standard input, so *tbl* may be used as a filter. When *tbl* is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

OPTIONS

−ms Copy the **−ms** macro package to the front of the output file.

−mm Copy the **−mm** macro package to the front of the output file.

EXAMPLE

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

```
         Household Population
       Town          Households
                   Number    Size
Bedminster            789    3.26
Bernards Twp.        3087    3.74
Bernardsville        2018    3.30
Bound Brook          3425    3.04
Branchburg           1644    3.49
Bridgewater          7897    3.81
Far Hills             240    3.19
```

SEE ALSO

troff(1), eqn(1)
Formatting Tables with tbl in
Editing and Text Processing on the Sun Workstation.

NAME

tcov – construct test coverage analysis and statement-by-statement profile

SYNOPSIS

tcov [**−a**] [−*n*] file.c ...

DESCRIPTION

Tcov produces a test coverage analysis and statement-by-statement profile of a C program. The coverage data for routines contained in a file named *file.c* is taken from the corresponding *file.d* produced by running the C compiler *cc*(1) with the **−a** option. An annotated listing of the program with coverage data is placed in *file.tcov*. Each straight-line segment of code (or each line if the **−a** option to *tcov* is specified) is prefixed with the number of times it has been executed; lines which have not been executed are prefixed with #####.

Note that the profile produced includes only the number of times each statement was executed, not execution times; to obtain times for routines use *gprof*(1) or *prof*(1).

Test coverage data from several runs will accumulate in the *.d* files.

OPTIONS

−a display an execution count for each statement; if **−a** is not specified, an execution count is displayed only for the first statement of each straight-line segment of code.

−*n* display table of the line numbers of the *n* most frequently executed statements and their execution counts.

SEE ALSO

cc(1), prof(1), gprof(1)

FILES

file.c	input C program file
file.d	input test coverage data file
file.tcov	output test coverage analysis listing file
/usr/bin/count	preprocessor for test coverage analysis
/usr/lib/bb_link.o	startoff and exit routines for test coverage analysis

BUGS

The analyzed program must call *exit*(2) or return normally for the coverage information to be saved in the *.d* file.

'Premature end of file' message for routine containing no statements.

NAME

tee – copy standard output to many files

SYNOPSIS

tee [**–i**] [**–a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*.

OPTIONS

–i Ignore interrupts.

–a Append the output to the *files* rather than overwriting them.

NAME

tektool – Tektronix 4014 terminal emulator tool

SYNOPSIS

tektool [–f *fontdir*] [**–s**[**lcdeg**[**ce**]] [**–c** *command line*] [**–r** *command line*]

DESCRIPTION

Tektool emulates a Tektronix 4014 terminal with the enhanced graphic module. It does this in much the same way as shelltool (see *suntools*(1)) emulates a regular glass tty. When **tektool** is invoked, a command (usually a shell) is started up, its output and input are connected to the emulator, and a new window is formed. The default window is the entire screen. When the emulator is running, buttons TF(1) through TF(4), (usually function keys 1-4 (see *kbd*(5)) have special meaning.

TF(1) Unshifted, this is the 4014 PAGE button. Shifted, this is the 4014 RESET button.

TF(2) Set local mode.

TF(3) Set on-line mode.

TF(4) Copy screen. The raster image (*/usr/include/rasterfile.h*) of the 4014 screen is piped to a command found in the TEKCOPY environment variable. The copy button is unaffected by window manipulations, and will transmit the contents of the 4014 screen only.

When in graphics input (GIN) mode and the 4014 crosshairs are visible, the left hand mouse button may be used as the space bar to terminate GIN mode.

OPTIONS

–f *fontdir*

Look for fonts in the directory specified by *fontdir*. The fonts must be called *tekfont*0 through *tekfont*3. Fonts must be in *vfont*(5) format. If this option is not given, the font directory is obtained from the TEKFONTS environment variable (if it exists). If no font directory is specified, */usr/lib/fonts* is used.

–s

Specifies the Tektronix 4014 strap options with the following modifiers:

l Received linefeeds also generate carriage returns.
c Received carriage returns also generate linefeeds.
d Received *DEL* characters are used as low order Y axis addresses.
e Echo keyboard input.
g Graphic input mode (GIN) terminator specification. If this strap is followed by a **c**, GIN mode data is terminated by a carriage return. If it is followed by a **e**, GIN mode data is terminated by a carriage return followed by an EOT character. If this strap is not present, no characters are sent after GIN mode data.

If the **–s** option is not given, the environment is searched for the TEKSTRAPS variable which provides the modifiers. If no straps are specified the **d** strap is assumed.

–c *command line*

Take terminal emulator input from a shell which in turn runs the *command line* following the **–c** option.

–r *command line*

Run *command line* to provide input to the terminal emulator. This must be the last option, since the remainder of the arguments are used.

CAVEATS

Like all 4014 emulators, this probably doesn't duplicate every nuance of the 4014. For instance, certain programs redraw stuff already on the screen in order to highlight things with the storage flash. Needless to say, this won't work here (however if you redraw it in writethru mode it will work!!). Also, even though the emulator supports the full 4096 address of the 4014, it cannot display this on the screen. All points will be rounded to the nearest available pixel. This may cause some funny effects.

The tektool window may be treated just like other windows; it can be overlaid, moved, reshaped etc. However, when the window is reshaped, the contents will not scale.

FILES

/usr/lib/fonts/tekfont[0-3]

SEE ALSO

suntools(1)
Tektronix 4014 and 4014-1 Computer Display Terminal User's Manual (070-1647-00)

BUGS

(1) Special point plot mode is not supported.

(2) Z axis stuff, except for defocusing, is not supported.

(3) Defocused alpha characters are not supported.

(4) The fonts need help.

(5) Needs scroll bars from the window system to make the whole 4014 screen accessable from a small window.

(6) There are no user prompts for waiting (for copy) and to remind people about the function keys.

(7) The current implementation seems to have an effective baud rate of between 4800 and 9600.

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

telnet [host [port]]

DESCRIPTION

Telnet communicates with another host using the TELNET protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt ("telnet>"). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an *open* command (see below) with those arguments.

Once a connection has been opened, *telnet* enters input mode. In this mode, text typed is sent to the remote host. To issue *telnet* commands when in input mode, precede them with the *telnet* "escape character" (initially "^]"). When in command mode, the normal terminal editing conventions are available.

TELNET COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed.

open *host* [*port*]
Open a connection to the named host. If the no port number is specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*(5)) or an Internet address specified in the "dot notation".

close Close a TELNET session and return to command mode.

quit Close any open TELNET session and exit *telnet*.

z Suspend *telnet*. This command only works when the user is using the *csh*(1).

escape [*escape-char*]
Set the *telnet* "escape character". Control characters may be specified as "^" followed by a single letter; e.g. "control-X" is "^X".

status Show the current status of *telnet*. This includes the peer one is connected to, as well as the state of debugging.

options
Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations will be displayed. Options sent by *telnet* are displayed as "SENT", while options received from the TELNET server are displayed as "RCVD".

crmod Toggle carriage return mode. When this mode is enabled any carriage return characters received from the remote host will be mapped into a carriage return and a line feed. This mode does not affect those characters typed by the user, only those received. This mode is not very useful, but is required for some hosts that like to ask the user to do local echoing.

? [*command*]
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information available about the command only.

SEE ALSO

rlogin(1C)

BUGS

There is no provision in the standard TELNET protocol to support ^S/^Q type commands. This implementation is very simple because *rlogin*(1C) is the standard mechanism used to communicate locally with hosts.

NAME

test − condition command

SYNOPSIS

test *expr*

DESCRIPTION

test evaluates the expression *expr*, and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. *test* returns a non zero exit if there are no arguments.

The following primitives are used to construct *expr*.

−b *file* true if the file exists and is a block special device.

−c *file* true if the file exists and is a character special device.

−d *file* true if the file exists exists and is a directory.

−f *file* true if the file exists and is not a directory.

−g *file* true if the file exists and is setgid.

−h *file* true if the file exists and is a symbolic link.

−k *file* true if the file exists and is sticky.

−l *string* the length of the string.

−n *s1* true if the length of the string *s1* is nonzero.

−r *file* true if the file exists and is readable.

−s *file* true if the file exists and has a size greater than zero.

−t [*fildes*]
true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

−w *file* true if the file exists and is writable.

−x *file* true if the file exists and is executable.

−z *s1* true if the length of string *s1* is zero.

s1 **=** *s2* true if the strings *s1* and *s2* are equal.

s1 **!=** *s2* true if the strings *s1* and *s2* are not equal.

s1 true if *s1* is not the null string.

n1 **−eq** *n2*
true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, or **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

! unary negation operator

−a binary *and* operator

−o binary *or* operator

(*expr* **)**
parentheses for grouping.

−a has higher precedence than **−o.** Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO

sh(1), find(1)

NAME

time – time a command

SYNOPSIS

time [command]

DESCRIPTION

There are two distinct versions of *time*: it is built in to the C-shell, and is an executable program available in */bin/time* when using the Bourne shell. In both cases, times are displayed on the diagnostic output stream.

In the case of the C-shell, a *time* command with no *command* argument simply displays a summary of time used by this shell and its children. When arguments are given the specified simple *command* is timed and the C-shell displays a time summary as described below under the *time* variable. If necessary, an extra shell is created to print the time statistic when the command completes.

The default resource-usage summary is a line of the form:

uuu.uu u *sss.ss* s *ee:ee pp*% *xxx*+*ddd*k *iii*+*ooo*io *mmm*pf+*ww*w

where *uuu.u* is the user time (U), *sss.s* is the system time (S), *ee*:*ee* is the elapsed time (E), *pp* is the percentage of CPU time versus elapsed time (P), *xxx* is the average shared memory in Kilobytes (X), *ddd* is the average unshared data space in Kilobytes (D), *iii* and *ooo* are the number of block input and output operations respectively (I and O), *mmm* is the number of page faults (M), and *ww* is the number of swaps (W).

The **time** variable controls the display that the C-shell prints when it times a command. The **time** variable can be supplied with one or two values. The first value is a number — *n* for instance. The C-shell displays a resource-usage summary for any command running for more than *n* CPU seconds. The second value is optional and is a character string which determines which resources the user wishes displayed. The character string can be any string of text with embedded control key-letters in it. A control key-letter is a percent sign (%) followed by a single *upper-case* letter. To print a percent sign, use two percent signs in a row. Unrecognized key-letters are simply printed. The control key-letters are:

D Average amount of unshared data space used in Kilobytes.
E Elapsed (wallclock) time for the command.
F Page faults.
I Number of block input operations.
K Average amount of unshared stack space used in Kilobytes.
M Maximum real memory used during execution of the process.
O Number of block output operations.
P Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time.
S Number of seconds of CPU time consumed by the kernel on behalf of the user's process.
U Number of seconds of CPU time devoted to the user's process.
W Number of swaps.
X Average amount of shared memory used in Kilobytes.

The *time* command in '/bin/time' times the given *command*, which must be specified, that is, *command* is not optional as it is in the C-shell's timing facility. When the command is complete, *time* displays the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

EXAMPLES

The two examples here show the differences between the *csh* version of *time* and the version in */bin/time*. The example assumes that *csh* is the shell in use.

```
angel% time wc /usr/man/man1/csh.1
    1876   11223   65895 /usr/man/man1/csh.1
```

```
2.7u 0.9s 0:03 91% 3+5k 19+2io 1pf+0w
angel% /bin/time wc /usr/man/man1/csh.1
   1876   11223   65895 /usr/man/man1/csh.1
        4.3 real         2.7 user         1.0 sys
angel%
```

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 50th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tip, cu − connect to a remote system

SYNOPSIS

tip [**−v**] [−*speed*] system-name
tip [**−v**] [−*speed*] phone-number
cu phone-number [**−t**] [**−s** *speed*] [**−a** *acu*] [**−l** *line*] [**−#**]

DESCRIPTION

Tip and *cu* establish a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote computer. It goes without saying that you must have a account on the machine (or equivalent) to which you wish to connect. The preferred interface is *tip*. The *cu* interface is included for those people attached to the 'call UNIX' command of the version 7 UNIX system. This manual page describes only *tip*.

When *tip* starts up it reads commands from the file *.tiprc* in your home directory. If you use the **−v** option on the *tip* command line, *tip* displays these commands as it executes them. See the discussion on *variables* later on.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well).

A tilde ('~') appearing as the first character of a line is an escape signal which directs *tip* to perform some special action. *Tip* recognizes the following escape sequences:

~^D ~.
Drop the connection and exit (you may still be logged in on the remote machine).

~c [*name*]
Change directory to *name* (no argument implies change to your home directory).

~! Escape to a shell (exiting the shell returns you to *tip*).

~> Copy file from local to remote.

~< Copy file from remote to local.

~p *from* [*to*]
Send a file to a remote UNIX host. When you use the put command, the remote UNIX system runs the command string
cat > *to*
while *tip* sends it the *from* file. If the *to* file isn't specified, the *from* file name is used. This command is actually a UNIX specific version of the '~>' command.

~t *from* [*to*]
Take a file from a remote UNIX host. As in the put command the *to* file defaults to the *from* file name if it isn't specified. The remote host executes the command string
cat > *from*; **echo ^A**
to send the file to *tip*.

~| Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.

~C Connect a program to the remote machine. The command string sent to the program is processed by the shell. The program inherits file descriptors 0 as remote line input, 1 as remote line output, and 2 as tty standard error.

~# Send a BREAK to the remote system. For systems which don't support the necessary *ioctl* call the break is simulated by a sequence of line speed changes and DEL characters.

~s Set a variable (see the discussion below).

~^Z Stop *tip* (only available when run under the C-Shell).

~? Get a summary of the tilde escapes

Copying files requires some cooperation on the part of the remote host. When a ~> or ~< escape is used to send a file, *tip* prompts for a file name (to be transmitted or received) and a command to be sent to the remote system, in case the file is being transferred from the remote system. The default end of transmission string for transferring a file from the local system to the remote is specified as the 'oe' parameter in the *remote*(5) file, but may be changed by the set command. While *tip* is transferring a file the number of lines transferred will be continuously displayed on the screen. A file transfer may be aborted with an interrupt. An example of the dialogue used to transfer files is given below (input typed by the user is shown in bold face).

```
arpa% tip monet
[connected]
...(assume we are talking to another UNIX system)...
ucbmonet login: sam
Password:
monet% cat > sylvester.c
~> Filename: sylvester.c
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~< Filename: reply.c
List command for remote host: cat reply.c
65 lines transferred in 2 minutes
monet%
...(or, equivalently)...
monet% ~p sylvester.c
...(actually echoes as ~[put] sylvester.c)...
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~t reply.c
...(actually echoes as ~[take] reply.c)...
65 lines transferred in 2 minutes
monet%
...(to print a file locally)...
monet% ~|Local command: pr -h sylvester.c | lpr
List command for remote host: cat sylvester.c
monet% ~^D
[EOT]
...(back on the local system)...
```

The *remote*(5) file contains the definitions for remote systems known by *tip*; refer to the remote manual page for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, for example:

```
tip —300 mds
```

When *tip* establishes a connection it sends out a connection message to the remote system. The default value for this string may be found in the remote file.

At any time that *tip* prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, aborts the dialogue and returns you to the remote machine.

When *tip* attempts to connect to a remote system, it opens the associated device with an exclusive-open *ioctl*(2) call. Thus only one user at a time may access a device. This is to prevent multiple processes from sampling the terminal line. In addition, *tip* honors the locking protocol used by *uucp*(1C).

AUTO-CALL UNITS

Tip may be used to dial up remote systems using a number of auto-call unit's (ACU's). When the remote system description contains the 'du' attribute, tip uses the call-unit ('cu'), ACU type ('at'), and phone numbers ('pn') supplied. Normally tip displays verbose messages as it dials. See *remote*(5) for details of the remote host specification.

Depending on the type of auto-dialer being used to establish a connection the remote host may have garbage characters sent to it upon connection. The user should never assume that the first characters typed to the foreign host are the first ones presented to it. The recommended practice is to immediately type a 'kill' character upon establishing a connection (most UNIX systems support '@' as the initial kill character).

Tip currently supports the Ventel MD-212+ autodialer modem and the Hayes SmartModem 1200.

REMOTE HOST DESCRIPTIONS

Descriptions of remote hosts are normally located in the system-wide file */etc/remote*. However, a user may maintain personal description files (and phone numbers) by defining and exporting the REMOTE shell variable. The *remote* file must be readable by *tip*, but a secondary file describing phone numbers may be maintained readable only by the user. This secondary phone number file is */etc/phones*, unless the shell variable PHONES is defined and exported. As described in *remote*(5), the *phones* file is read when the host description's phone number(s) capability is an '@'. The phone number file contains lines of the form:

```
system-name    phone-number
```

Each phone number found for a system is tried until either a connection is established, or an end of file is reached. Phone numbers are constructed from '0123456789-=*', where the '=' and '*' are used to indicate a second dial tone should be waited for (ACU dependent).

TIP INTERNAL VARIABLES

Tip maintains a set of variables which are used in normal operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the '~s' escape. The syntax for variables is patterned after *vi*(1) and *mail*(1). Supplying 'all' as an argument to the set command displays all variables that the user can read. Alternatively, the user may request display of a particular variable by attaching a '?' to the end. For example 'escape?' displays the current escape character.

Variables are numeric, string, character, or Boolean values. Boolean variables are set merely by specifying their name. They may be reset by prepending a '!' to the name. Other variable types are set by appending an '=' and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables.

Variables may be initialized at run time by placing set commands (without the '~s' prefix) in a *.tiprc* file in one's home directory. The **−v** option makes *tip* display the sets as they are made. Comments preceded by a '#' sign can appear in the *.tiprc* file.

Finally, the variable names must either be completely specified or an abbreviation may be given. The following list details those variables known to *tip*, their abbreviations (surrounded by brackets), and their default values. Those variables initialized from the remote file are marked with a '*'. A mode is given for each variable — capitalization indicates the read or write capability is given only to the super-user.

Variable	**Type**	**Mode**	**Default**	**Description**
[be]autify	bool	rw	true	discard unprintables when scripting

]ba]udrate	num	rW	*	connection baud rate
[c]har[delay]	num	rw	0	character delay for file transfers to remote (cl)
[dial]timeout	num	rW	60	timeout (seconds) when establishing connection
]di]sconnect	str	rw	""	string to send to disconnect (di)
[ec]hocheck	bool	rw	false	
]eofr]ead	str	rw	*	char's signifying EOT from the remote host
]eofw]rite	str	rw	*	string sent for EOT
[eol[	str	rw	*	end of line indicators
]es]cape	char	rw	‘~’	command prefix character
[et[imeout	num	rw	10	echo check timeout (et)
]ex]ceptions	str	rw	"\t\n\f\b"	char's not discarded due to beautification
[fo]rce	char	rw	’’	force character
[fr[amesize	num	rw	*	size of buffering between writes on reception
[h]alf]d]uple]x]		bool	rw	falsehost is half duplex — do local echo (hd)
[ho]st	str	r	*	name of host connected to
[l]ine[delay]	num	rw	0	line delay for transfers to remote (dl)
[l]ocal]e]cho	bool	rw	false	synonym for halfduplex
]lock]	str	RW	"/tmp/aculock"	lock file for ACU logging
[log]	str	RW	"/usr/adm/aculog"	ACU log file
[par[ity	str	rw	"none"	parity to be generated (pa)
]phones]	str	r	"/etc/phones"	file for hidden phone numbers
[pr[ompt	char	rW	‘\n’	end of line indicator set by host
[ra]ise	bool	rw	false	upper case mapping switch
]r]aise[c]har	char	rw	’’	interactive toggle for raise
[raw]ftp	bool	rw	false	send all characters during file transfer (rw) do not filter non-printable characters do not do translations like \n to \r.
[rec[ord	str	rw	"tip.record"	name of script output file
[remote]	str	r	"/etc/remote"	system description file
[sc[ript	bool	rw	false	session scripting switch
]tab[expand	bool	rw	false	expand tabs during file transfers
]ta[ndem	bool	rw	true	use XON/XOFF flow control (ta and nt)
]verb]ose	bool	rw	true	make noise during file transfers
]SHELL[	str	rw	"/bin/sh"	name of shell for ~! escape
]HOME[	str	rw	""	home directory for ~c escape

ENVIRONMENT VARIABLES

The following variables are read from the environment:

REMOTE	The location of the *remote* file.
PHONES	The location of the file containing private phone numbers.
HOST	A default host to connect to.
HOME	One's log-in directory (for chdirs).
SHELL	The shell to fork on a ‘~!’ escape.

FILES

~/.tiprc initialization file.

/usr/spool/uucp/LCK..*	lock file to avoid conflicts with *uucp*

DIAGNOSTICS

Diagnostics are, hopefully, self explanatory.

SEE ALSO

remote(5), phones(5)

BUGS

Note that *chardelay* and *linedelay* are currently not implemented.

NAME

touch – update date last modified of a file

SYNOPSIS

touch [**–c**] [**–f**] file **...**

DESCRIPTION

Touch attempts to set the modified date of each *file*. If the file exists, this is done by reading a character from the file and writing it back.

Touch is valuable when used in conjunction with *make*(1), where, for instance, you might want to force a complete rebuild of a program composed of many pieces. In such a case, you might type, for example:

```
% touch *.c
% make
```

and the *make* would then see that all the **.**c files were more up to date than all the corresponding **.**o files, and would start the build from scratch.

OPTIONS

–c Do not attempt to create a *file* if it does not exist.

–f Attempt to force the touch in spite of read and write permissions on a *file*.

SEE ALSO

utimes(2)

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. The arguments *string1* and *string2* are considered sets of characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character.

In either string the notation *a−b* means a range of characters from *a* to *b* in increasing ASCII order. The character '\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A '\' followed by any other character stands for that character.

OPTIONS

Any combination of the options **−cds** may be used:

−c Complement the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal;

−d Delete all input characters in *string1;*

−s Squeeze all strings of repeated output characters that are in *string2* to single characters.

EXAMPLE

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr −cs A−Za−z ´\012´ <file1 >file2
```

SEE ALSO

ed(1), ascii(7), expand(1)

BUGS

Won't handle ASCII NUL in *string1* or *string2;* always deletes NUL from input.

NAME

troff — typeset or format documents

SYNOPSIS

troff [**−o***pagelist*] [**−n***N*] [**−s***N*] [**−m** *name*] [**−r***aN*] [**−i**] [**−q**] [**−t**] [**−f**]
[**−w**] [**−b**] [**−a**] [**−p***N*] [file] **...**

DESCRIPTION

Troff formats text in the named *files*. The output is by default destined for printing on a Graphic Systems C/A/T phototypesetter, but suitable postprocessing software can convert the C/A/T output to a form which can be directed to other high-resolution devices. See also the *nroff*(1) manual page, which describes a formatter which formats text for typewriter-like devices. The capabilities of both *troff* and *nroff* are described in *Formatting Documents with Nroff and Troff*.

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (**−**) is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear before the *files*.

−o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N−M* means pages *N* through *M*; an initial *−N* means from the beginning to page *N*; and a final *N−* means from *N* to the end.

−n*N* Number first generated page *N*.

−m*name*
Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

−t Direct output to the standard output instead of the phototypesetter. In general, you will have to use this option if you don't have a typesetter attached to the system.

−a Send a printable ASCII approximation of the results to the standard output.

Some options of *troff* only apply if you have a C/A/T typesetter attached to your system. These options are here for historical reasons:

−s*N* Stop every *N* pages. *Troff* stops the phototypesetter every *N* pages, produces a trailer to allow changing cassettes, and resumes when the typesetter's start button is pressed.

−f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

−w Wait until phototypesetter is available, if currently busy.

−b Report whether the phototypesetter is busy or available. No text processing is done.

−pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elasped time.

If the file */usr/adm/tracct* is writable, *troff* keeps phototypesetter accounting records there. The integrity of that file may be secured by making *troff* a 'set user-id' program.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*
/dev/cat	phototypesetter

/usr/adm/tracct accounting statistics for /dev/cat

SEE ALSO

Formatting Documents with Nroff and Troff in
Editing and Text Processing on the Sun Workstation
nroff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1), checknr(1)

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

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

EXAMPLE

```
while true
do
    command list
done
```

SEE ALSO

csh(1), sh(1), false(1)

DIAGNOSTICS

True has exit status zero.

NAME

tset − establish terminal characteristics for the environment

SYNOPSIS

tset [**−**] [**−e***c*] [**−I**] [**−k***c*] [**−n**] [**−Q**] [**−r**] [**−s**] [**−S**]
  [**−m** [*port-ID*[*baudrate*][*:type*] ...] [*type*]

reset ...

DESCRIPTION

Tset sets up your terminal, typically when you first log in. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. *Tset* first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each UNIX port is specified in the */etc/ttytype* database. Type names for terminals may be found in the *termcap*(5) database. If a port is not wired permanently to a specific terminal (not hardwired) it is given an appropriate generic identifier such as *dialup*.

Reset turns off cbreak and raw modes, output delays, and parity checking; turns on newline translation, echo, and tab expansion; and restores undefined special characters to their default state. Then it sets the modes as usual, based on the terminal type (which will probably override some of the above). (See *stty*(1) for more information.) *Reset* also uses the *rs=* and *rf=* "reset string and file" instead of the initialization string and file from */etc/termcap*. This is useful after a program dies and leaves the terminal in a funny state. Often in this situation, characters will not echo as you type them. You may have to type "<LF>reset<LF>" since <CR> may not work.

When no arguments are specified, *tset* simply reads the terminal type out of the TERM environment variable and re-initializes the terminal. *Tset* also performs mode and environment initialization — typically done once at login — and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script (*.profile* for *sh*(1) users or *.login* for *csh*(1) users) it is desirable to give information about the type of terminal you will usually use on ports that are not hardwired. These ports are identified in */etc/ttytype* as *dialup* or *plugboard*, etc. Any of the alternate generic names given in *termcap* may be used for the identifier. Refer to the **−m** option under OPTIONS for more information. If no mapping applies and a final *type* option, not preceded by a **−m**, is given on the command line then that type is used; otherwise the identifier found in the */etc/ttytype* database is used as the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by *tset*, and information about the terminal's capabilities to a shell's environment. This can be done using the **−**, **-s**, or **-S** options. Refer to OPTIONS for more information. *sh*(1):

```
export TERM; TERM=`tset − options...`
```

or using the C shell, *csh*(1):

```
setenv TERM `tset − options...`
```

With *csh* it is convenient to make an alias in your .cshrc:

```
alias tset ´setenv TERM `tset − \!*` ´
```

Either of these aliases allow the command

```
tset 2621
```

to be invoked at any time from your login csh. **Note to Bourne Shell users:** It is **not** possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. If a process could set its parent's environment, none of this nonsense would be necessary in the first place.

Once the terminal type is known, *tset* engages in terminal driver mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and newline expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is '#', the erase character is changed as if **-e** had been used.

OPTIONS

— The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the TERM environment variable.

—e*c* Set the erase character to be the named character *c* on all terminals. Default is the backspace key on the keyboard, usually ^H. The character *c* can either be typed directly, or entered using the hat notation used here.

—I Suppress transmitting terminal-initialization strings.

—k*c* Set the line kill character to be the named character *c* on all terminals. Default is ^U. The kill character is left alone if **—k** is not specified. The hat notation can also be used for this option.

—n Specifies that the new tty driver modes should be initialized for this terminal. Probably useless since **stty new** is the default.

—Q Suppress printing the "Erase set to" and "Kill set to" messages.

—r In addition to other actions, reports the terminal type.

—s Output *setenv* commands for TERM and TERMCAP. This can be used with

```
set noglob
eval `tset —s ...`
unset noglob
```

and is preferred to `setenv TERM `tset – . . .`', because it makes programs such as *vi*(1) start up much faster.

—S Similar to the —s option, but outputs two strings suitable for use in *csh*(1) *.login* files as follows:

```
set noglob
set t=(`tset —S ...`)
setenv TERM $t[1]
setenv TERMCAP "$t[2]"
unset t
unset noglob
```

—m [*port-ID*[*baudrate*]:*type*] ...
Specify ("map") a terminal type when connected to a generic port (such as *dialup* or *plugboard*) identified by *port-ID*. The *baudrate* argument (see also, *stty*(1)) can be used to check the baudrate of the port and set the terminal type accordingly. The target rate is prefixed by any combination of the following operators:

>	is greater than
@	equals or "at"
<	is less than
!	it is not the case that (negates the above operators)

to specify the conditions under which the mapping is made.

In the following example, the terminal type is set to *adm3a* if the port is a dialup with a speed of greater than 300 or to *dw2* if the port is a dialup at 300 baud or less. In the third case, the question mark preceeding the terminal type indicates that the user is to verify the type desired. A null response indicates that the named type is correct. Otherwise, the user's response is taken to be the type desired.

```
tset −m 'dialup>300:adm3a' −m 'dialup:dw2' −m 'plugboard:?adm3a'
```

To prevent interpratation as metacharacters, the entire argument to **−m** should be enclosed in single quotes. When using the C-shell, exclamation points should be preceded by a backslash (\).

EXAMPLES

These examples all assume the C-shell and use the −s option. If you use the Bourne shell, use one of the variations described above. Note that a typical use of *tset* in a .profile or .login will also use the **−e** and **−k** options, and often the **−n** or **−Q** options as well. These options have not been included here to keep the examples small.

At the moment, you are on a 2621. This is suitable for typing by hand but not for a .profile, unless you are *always* on a 2621.

```
set noglob; eval `tset −s 2621`; unset noglob
```

You have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttytype.

```
set noglob; eval `tset −s −m dialup:h19`; unset noglob
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use a vt100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone else's terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
set noglob; eval `tset −s −m 'switch>1200:?vt100' −m 'switch<=1200:2621'`; unset
noglob
```

All of the above entries will fall back on the terminal type specified in */etc/ttytype* if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
set noglob; eval `tset −s ?adm3a`; unset noglob
```

If the file */etc/ttytype* is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
set noglob; eval `tset −s −m '>1200:vt100' 2621`; unset noglob
```

Here is a fancy example to illustrate the power of *tset* and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in */etc/ttytype*. You want your erase character set to control H, your kill character set to control U, and don't want *tset* to print the "Erase set to Backspace, Kill set to Control U" message. This example appears to take up more than one line. When you type in real *tset* commands, you must enter them entirely on one line.

```
set noglob; eval `tset −e −k^U −Q −s −m 'switch<=1200:concept100' −m
'switch:?vt100' −m dialup:concept100 −m arpanet:dm2500`; unset noglob
```

FILES

/etc/ttytype	port name to terminal type mapping database
/etc/termcap	terminal capability database

/usr/lib/tabset/* tab setting sequences for various terminals. Pointed to by *termcap* entries.

SEE ALSO

csh(1), sh(1), stty(1), ttytype(5), termcap(5), environ(5)

BUGS

The *tset* command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the *login*(1) program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

It could well be argued that the shell should be responsible for ensuring that the terminal remains in a sane state; this would eliminate the need for the *reset* program.

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty – get terminal name

SYNOPSIS

tty [**−s**]

DESCRIPTION

Tty prints the pathname of the user's terminal unless the **−s** (silent) option is given. In either case, the exit value is zero if the standard input is a terminal, and one if it is not.

NAME

ul – do underlining

SYNOPSIS

ul [**–i**] [**–t** *terminal*] [file ...]

DESCRIPTION

Ul reads the named *files* (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. *ul* uses the */etc/termcap* file to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

OPTIONS

–t Override the terminal kind specified in the environment. If the terminal cannot underline, underlining is ignored.

–i Indicate underlining onto by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

BUGS

Nroff usually generates a series of backspaces and underlines intermixed with the text to indicate underlining. *Ul* makes attempt to optimize the backward motion.

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

/usr/sccs/unget [**−r** *SID*] [**−s**] [**−n**] file ...

DESCRIPTION

Unget undoes the effect of a **get −e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

OPTIONS

Options apply independently to each named file.

−r *SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The **−r** option is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary but omitted from the command line.

−s Suppress displaying the intended delta's *SID*.

−n Retain the gotten file — it is normally removed from the current directory.

SEE ALSO

sccs(1), delta(1), get(1), sact(1).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**−udc** [+n] [−n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see *sort*(1).

OPTIONS

−u Copy only those lines which are *not* repeated in the original file.

−d Write one copy of just the repeated lines.

The normal output of *uniq* is the union of the **−u** and **−d** options

−c supersedes **−u** and **−d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.54000e+00
        / 3.93701e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 pounds force/in2
You want: atm
        * 1.02069e+00
        / 9.79730e-01
```

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

BUGS

Don't base your financial plans on the currency conversions.

NAME

uptime — show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w*(1) command.

EXAMPLE

```
angel% uptime
  6:47am  up 6 days, 16:38,  1 users,  load average: 0.69, 0.28, 0.17
angel%
```

FILES

/vmunix	system name list

SEE ALSO

w(1)

NAME

users – compact list of users who are on the system

SYNOPSIS

users

DESCRIPTION

Users lists the login names of the users currently on the system in a compact, one-line format:

```
% users
john paul george
%
```

FILES

/etc/utmp

SEE ALSO

who(1)

NAME

uucp, uulog – unix to unix copy

SYNOPSIS

uucp [**–C**] [**–c**] [**–d**] [**–m**] *source_file ... destination_file*
uulog [**–s***sys*] [**–u***user*]

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form

system-name!pathname

where 'system-name' is taken from a list of system names which *uucp* knows about. Shell metacharacters ? * [ ] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ˜*user*; where *user* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

Uulog maintains a summary log of *uucp* and *uux*(1C) transactions in the file */usr/spool/uucp/LOGFILE*, by gathering information from partial log files named */usr/spool/uucp/LOG.*.?*. It removes the partial log files.

OPTIONS

The following options are interpreted by *uucp*.

–C Make a copy of outgoing files in the uucp spool directory, rather than copying the source file directly to the target system. This lets you remove the source file after issuing the uucp command.

–c Use the source file when copying out rather than copying the file to the spool directory. This is the default.

–d Make all necessary directories for the file copy.

–m Send mail to the requester when the copy is complete.

UULOG OPTIONS

–s*sys* Print information about work involving system *sys*.

–u*user*
Print information about work done for the specified *user*.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/*	other data and program files

SEE ALSO

uux(1C), mail(1)
Uucp Implementation Description in the *Sun System Manager's Manual.*

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

BUGS

All files received by *uucp* will be owned by *uucp*.
The **−m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [ ] will not activate the **−m** option.

NAME

uuencode,uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [source-file] destination-file ¦ **mail** sys1!sys2!...!decode
uudecode [source-file]

DESCRIPTION

Uuencode and *uudecode* are used to send a binary file via uucp (or other) mail. This combination of commands can be used over indirect mail links even when *uusend*(1C) is not available.

Uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

Uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user "decode" should be filtered through the *uudecode* program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using *sendmail* or by making *rmail* be a link to *Mail* instead of *mail*. In each case, an alias must be created in a master file to get the automatic invocation of *uudecode*.

If these facilities are not available, the file can be sent to a user on the remote machine who can *uudecode* it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

SEE ALSO

uuencode(5), uusend(1C), uucp(1C), uux(1C), mail(1)

BUGS

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking *uudecode* (often *uucp)* must have write permission on the specified file.

NAME

uusend – send a file to a remote host

SYNOPSIS

uusend [**−m** mode] sourcefile sys1!sys2!..!remotefile

DESCRIPTION

Uusend sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of *uucp*(1C) links needs to connect the two systems.

The sourcefile can be "–", meaning to use the standard input. Both of these options are primarily intended for internal use of *uusend*.

The remotefile can include the ~userid syntax.

OPTIONS

−m *mode*

Take the mode of the file on the remote end from the octal number specified as *mode*. The mode of the input file is used if the **−m** option is not specified.

DIAGNOSTICS

If anything goes wrong any further away than the first system down the line, you will never hear about it.

SEE ALSO

uux(1C), uucp(1C), uuencode(1C)

BUGS

This command shouldn't exist, since *uucp* should handle it.

All systems along the line must have the *uusend* command available and allow remote execution of it.

Some UUCP systems have a bug where binary files cannot be the input to a *uux* command. If this bug exists in any system along the line, the file will show up severely munged.

NAME

uux − unix to unix command execution

SYNOPSIS

uux [**−**] [**−r**] [**−x*n***] [**−z**] [**−g*x***] *command-string*

DESCRIPTION

Uux will gather 0 or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by system-name!. A null system-name is interpreted as the local system.

File names may be one of

(1) a full pathname;

(2) a pathname preceded by ~*xxx*; where *xxx* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

The '−' option will cause the standard input to the *uux* command to be the standard input to the command-string.

For example, the command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"
```

will get the f1 files from the usg and pwba machines, execute a *diff* command and put the results in f1.diff in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

OPTIONS

−r don't start *uucico*, just queue the job.

−x*n* set debugging level to *n*.

−n don't return any indication by *mail* of success or failure of the job.

−z return an indication by *mail* only if the job fails.

−g*x* set service grade or classification to *x*. The default is **A**.

FILES

/usr/spool/uucp spool directory
/usr/lib/uucp/*other data and programs

SEE ALSO

uucp(1C)
D. A. Nowitz, *Uucp Implementation Description*

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from *uux*. Typically, a restricted site will permit little other than the receipt of mail via *uux*.

BUGS

Only the first command of a shell pipeline may have a system-name!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do.
The shell tokens << and >> are not implemented.
There is no notification of denial of execution on the remote machine.

NAME

val − validate SCCS file

SYNOPSIS

/usr/sccs/val −
/usr/sccs/val [**−s**] [**−r** *SID*] [**−m** *name*] [**−y** *type*] file ...

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order.

Val has a special argument, −, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

OPTIONS

Options apply independently to each named file on the command line.

−s Silence diagnostic messages normally generated for errors detected while processing the specified files.

−r *SID* The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (for instance, **r1** is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (for instance, **r1.0** or **r1.1.0** are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

−m *name*
name is compared with the SCCS %M% keyword in *file*.

−y *type*
type is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, that is, can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate option;
bit 2 = corrupted SCCS file;
bit 3 = can't open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **−y** mismatch;
bit 7 = %M%, **−m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned — logical OR of the codes generated for each command line and file processed.

SEE ALSO

sccs(1), admin(1), delta(1), get(1), prs(1).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a

core dump.

NAME

vax − is current machine a vax

SYNOPSIS

if vax; then ...; fi *(sh)*

```
if { vax } then          (csh)
   ...
endif
```

DESCRIPTION

The *vax* command is, on VAX'en, the same as *true*(1); on Sun Workstations and other machines it is the same as *false*(1).

SEE ALSO

false(1), sun(1), true(1)

NAME

vfontinfo – inspect and print out information about UNIX fonts

SYNOPSIS

vfontinfo [**–v**] fontname [characters]

DESCRIPTION

Vfontinfo allows you to examine a font in the UNIX format. It prints out all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format.

The *fontname* argument is the name of the font you wish to inspect. It writes to standard output. If it can't find the file in your working directory, it looks in */usr/lib/vfont* (the place most of the fonts are kept).

The *characters*, if given, specify certain characters to show. If omitted, the entire font is shown.

If the **–v** (verbose) flag is used, the bits of the glyph itself are shown as an array of X's and spaces, in addition to the header information.

SEE ALSO

vpr(1), vfont(5)
The Berkeley Font Catalog

NAME

vgrind − grind nice listings of programs

SYNOPSIS

vgrind [**−f**] [**−**] [**−t**] [**−n**] [**−x**] [**−W**] [**−sn**] [**−h** header] [**−d** file] [**−l***language*] *filename* ®.IX "vgrind command" "" "**vgrind** — make formatted C listings"

DESCRIPTION

Vgrind formats the program sources which are *file* arguments in a nice style using *troff*(1). Comments are placed in italics, keywords in bold face, and the name of the current function is listed down the margin of each page as it is encountered.

Vgrind runs in two basic modes, filter mode or regular mode. In filter mode *vgrind* acts as a filter in a manner similar to *tbl*(1). The standard input is passed directly to the standard output except for lines bracketed by the *troff*-like macros:

.vS - starts processing

.vE - ends processing

These lines are formatted as described above. The output from this filter can be passed to *troff* for output. There need be no particular ordering with *eqn*(1) or *tbl*(1).

In regular mode *vgrind* accepts input *file*s, processes them, and passes them to *troff*(1) for output.

In both modes *vgrind* passes any lines beginning with a decimal point without conversion.

OPTIONS

−f force filter mode.

− take from standard input (default if **−f** is specified).

−t similar to the same option in *troff*, that is, formatted text goes to the standard output.

−n do not make keywords boldface.

−x output the index file in a 'pretty' format. The index file itself is produced whenever *vgrind* is run with a file called *index* in the current directory. The index of function definitions can then be run off by giving *vgrind* the **−x** option and the file *index* as argument.

−W force output to the (wide) Versatec printer rather than the (narrow) Varian.

−s specifies a point size to use on output (exactly the same as the argument of a *troff* **.**ps (point size) request.

−h specifies a particular header to put on every output page (default is the file name).

−d specifies an alternate language definitions file (default is */usr/lib/vgrindefs*).

−l specifies the language to use. Currently known are C (**−lc** the default), CSH (**−lcsh**), ICON (**−lI**), ISP (**−i**), LDL (**−lLDL**), MODEL (**−lm**), PASCAL (**−lp**), RATFOR (**−lr**), and SHELL (**−lsh**).

FILES

index	file where source for index is created
/usr/lib/tmac/tmac.vgrind	macro package
/usr/lib/vfontedpr	preprocessor
/usr/lib/vgrindefs	language descriptions

BUGS

Vfontedpr assumes that a certain programming style is followed:

For C — function names can be preceded on a line only by spaces, tabs, or an asterisk. The parenthesized arguments must also be on the same line.

For PASCAL — function names need to appear on the same line as the keywords *function* or *procedure*.

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of spaces to align source code fails miserably; if you plan to *vgrind* your program you should use tabs. This is somewhat inevitable since the font used by *vgrind* is variable width.

The mechanism of *ctags*(1) in recognizing functions should be used here.

NAME

vi – screen oriented (visual) display editor based on ex

SYNOPSIS

vi [**–R**] [**–r**] [**–t** *tag*] [**+***command*] [**–x**] [**–w***nnn*] [**–l**] *file* ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*(1). *Ex* and *vi* are in fact the same text editor; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

OPTIONS

–R edit the file in read-only state. You can achieve the same effect with the *view* command.

–r recover the indicated *files* after a crash.

–t *tag* edit the file containing the tag *tag*. A tags database must have been built previously using the *ctags*(1) command.

+*command*
start the editing session by executing *command*.

–w*nnn*
set the default window (number of lines on your terminal) to *nnn*— this is useful if you are dialling into the system over a slow 'phone line.

–x prompt for a key to be used in encrypting the file being edited.

–l set up for editing LISP programs.

FILES

See *ex*(1).

SEE ALSO

ex (1), edit (1), "Vi Quick Reference" card,
Using vi, the Visual Display Editor in
Editing and Text Processing on the Sun Workstation.

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.

Repeating a change which wraps over the margin when *wrapmargin* is in effect doesn't generally work well: sometimes it just makes a mess of the change, and sometimes even leaves you in insert mode. A way to work around the problem is to replicate the changes using yank and put.

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*.

When using the **–r** option to recover a file, you must write the recovered text before quitting or you will lose it. *Vi* does not prevent you from exiting without writing unless you make changes.

RESTRICTIONS

The encryption facilities of *vi* are not available on software shipped outside the U.S.

NAME

view – view a file without changing it using the vi visual editor

SYNOPSIS

view [**−t** tag] [**−r**] [**+***command*] [**−l**] [**−w***n*] name ...

DESCRIPTION

View uses the *vi* (visual) or display oriented text editor to browse through a file interactively without actually making any changes to the file. It is possible to get to the command mode of *ex* from within *view* and vice-versa, just as when using *vi*.

FILES

See *ex*(1).

SEE ALSO

ex (1), edit (1), vi(1), "Vi Quick Reference" card,
Using vi, the Visual Display Editor in
Editing and Text Processing on the Sun Workstation.

NAME

vlp – Format Lisp programs to be printed with nroff, vtroff, or troff

SYNOPSIS

vlp [**−p** *pointsize*] [**−d**] [**−f**] [**−l**] [**−v**] [**−T** *title1*] file1 [**−T** *title2*] file2 ...

DESCRIPTION

Vlp formats the named files so that they can be run through nroff, vtroff, or troff to produce listings that line-up and are attractive. The first non-blank character of each line is lined-up vertically, as in the source file. Comments (text beginning with a semicolon) are printed in italics. Each function's name is printed in bold face next to the function. This format makes Lisp code look attractive when it is printed with a variable width font.

Normally, *vlp* works as a filter and sends its output to the standard output. However, the **−v** switch pipes the output directly to vtroff. If no files are specified, then *vlp* reads from the standard input.

The following options are available:

−p The **−p** switch changes the size of the text from its default value of 8 points to one of 6, 8, 10, or 12 points. Once set, the point size is used for all subsequent files. This point size does not apply to embedded text (see *−f* below).

−d The **−d** switch puts *vlp* into debugging mode.

−f *Vlp* has a filtered mode in which all lines are passed unmodified, except those lines between the directives **.Ls** and **.Le**. This mode can be used to format Lisp code that is embedded in a document. The directive **.Ls** takes an optional argument that gives the point size for the embedded code. If not size is specified, the size of the surrounding text is used.

−l The **−l** switch prevents *vlp* from placing labels next to functions. This switch is useful for embedded Lisp code, where the labels would be distracting.

−v This switch cause *vlp* to send its output to vtroff rather than the standard output.

−T A title to be printed on each page may be specified by using the **−T** switch. The **−T** switch applies only to the next file name given. Titles are not printed for embedded text (see **−f**, above). This switch may not be used if *vlp* is reading from the standard input.

FILES

/usr/lib/vlpmacs troff/nroff macros

SEE ALSO

vgrind(1), lisp(1)

BUGS

vlp transforms \ into \\ so that it will be printed out. Hence, troff commands cannot be embedded in Lisp code.

NAME

vplot – plot graphics on the Versatec

SYNOPSIS

vplot [**–W**] [**–V**] [**–b** *lpr-arg*] file

DESCRIPTION

Vplot reads *plot*(5) format graphics input from the file specified by *file* (standard input if no *file* is specified) and produces a plot on the Varian or Versatec.

OPTIONS

–W force output to the (wide) Versatec printer rather than the standard Versatec printer.

–V force output to the standard Versatec printer.

–b *lpr-arg*
arg (the next argument on the command line) specifies extra arguments to *lpr*(1).

SEE ALSO

plot(1G), lpr(1), plot(5)

NAME

vpr, vprm, vpq, vprint – raster printer/plotter spooler

SYNOPSIS

vpr [**–W**] [**–l**] [**–v**] [**–t** [**–1234** font]] [**–w**] [**–w***width*] [**–m**] [name ...]
vprm [id ...] [filename ...] [owner ...]
vpq
vprint [**–W**] file ...

DESCRIPTION

Vpr causes the named files to be queued for printing or typeset simulation on one of the available raster printer/plotters. If no files are named, the standard input is read. By default the input is assumed to be line printer-like text. For very wide plotters, the input is run through the filter */usr/lib/sidebyside* giving it an argument of **–w106** which arranges it four pages adjacent with 90 column lines (the rest is for the left margin). Since there are 8 lines per inch in the default printer font, *vpr* thus produces 86 lines per page (the top and bottom lines are left blank).

The following options are available:

–l Print the input in a more literal manner. Page breaks are not inserted, and most control characters (except format effectors: \n, \f, etc.) are printed (many control characters print special graphics not in the ASCII character set.) Tab and underline processing is still done. If this option is not given, control characters which are not format effectors are ignored, and page breaks are inserted after an appropriate number of lines have been printed on a page.

–W Queues files for printing on a wide output device, if available. Normally, files are queued for printing on a narrow output device.

–1234 Specifies a font to be mounted on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size.*

–m Report by *mail*(1) when printing is complete.

–w (Applicable only to wide output devices.) Do not run the input through sidebyside. Such processing has been done already, or full (440 character) printer width is desired.

–w*width* Use width *width* rather than 90 for *sidebyside.*

–v Use the filter */usr/lib/vrast* to convert the vectors to raster. The named files must be a parameter and vector file (in that order) created by *plot*(3X) routines.

–t Use the filter */usr/lib/vcat* to typeset the input on the printer/plotter. The input must have been generated by *troff*(1) run with the **–t** option. This is not normally run directly to wide output devices, since it is wasteful to run only one page across. The program *vtroff*(1) is normally used and arranges, using *vsort* for printing to occur four pages across, conserving paper.

Vprm removes entries from the raster device queues. The id, filename or owner should be that reported by *vpq*. All appropriate files will be removed. Both queues are always searched. The id of each file removed from the queue will be printed.

Vpq prints the queues. Each entry in the queue is printed showing the owner of the queue entry, an identification number, the size of the entry in characters, and the file which is to be printed. The *id* is useful for removing a specific entry from the printer queue using *vprm*

Vprint is a shell script which *pr's* a copy of each named file on one of the electrostatic printer/plotters. The files are normally printed on a narrow device; **–W** option causes them to be printed on a wide device.

FILES

/usr/spool/v?d/*	device spool areas
/usr/lib/v?d	daemons
/usr/lib/vpd	Versatec daemon
/usr/lib/vpf	filter for printer simulation
/usr/lib/*vcat	filter for typeset simulation
/usr/lib/vrast	filter for plot
/usr/lib/sidebyside	filter for wide output

SEE ALSO

troff(1), vfont(5), vp(4), pti(1), vtroff(1), plot(3X)

BUGS

The 1's (one's) and l's (lower-case el's) in a Benson-Varian's standard character set look very similar; caution is advised.

A versatec's hardware character set is rather ugly. *Vprint* should use one of the constant width fonts to produce prettier listings.

NAME

vswap – convert a foreign font file

SYNOPSIS

/usr/lib/vswap [**–r**]

DESCRIPTION

Without the **-r** option, *vswap* translates its standard input (which must be a *vfont* file in the reversed-byte order) into a locally correct *vfont* file on its standard output. With the **-r** option, *vswap* translates its standard input (which must be a *vfont* file in the local-byte order) into a byte-reversed *vfont* file on its standard output.

The UNIX *vfont* representation for fonts is a binary file containing machine-dependent elements — short (16-bit) integers, in particular. There are (at least) two common ways of representing a 16-bit integer. A program compiled on a VAX will expect the VAX format, while the same program compiled on a Motorola 68000 will expect 68000 format. *Vswap* can be used to convert font files created on a VAX to the format required to use them on the Sun. It can also convert Sun-format font files to VAX format (with the **–r** option). Since the font files are in the byte order of the local machine, programs which access the font files don't need to be concerned with byte-swapping issues. This could be considered a bug.

SEE ALSO

troff(1), vfont(5)

BUGS

A machine-independent font format should be defined.

NAME

vtroff – troff to a raster plotter

SYNOPSIS

vtroff [**–w**] [**–F** majorfont] [**–123** minorfont] [**–l***length*] [**–x**] troff arguments

DESCRIPTION

Vtroff runs *troff*(1) sending its output through various programs to produce typeset output on a raster plotter such as a Benson-Varian or or a Versatec. The **–W** option specifies that a wide output device be used; the default is to use a narrow device. The **–l** (lower case l) option causes the output to be split onto successive pages every *length* inches rather than the default 11".

The default font is a Hershey font. If some other font is desired you can give a **–F** argument and then the font name. This will place normal, italic and bold versions of the font on positions 1, 2, and 3. To place a font only on a single position, you can give an argument of the form *–n* and the minor font name. A **.r** will be added to the minor font name if needed. Thus "vtroff –ms paper" will set a paper in the Hershey font, while "vtroff –F nonie –ms paper" will set the paper in the (sans serif) nonie font. The **–x** option asks for exact simulation of photo-typesetter output. (I.e. using the width tables for the C.A.T. photo-typesetter)

FILES

/usr/lib/tmac/tmac.vcat	default font mounts and bug fixes
/usr/lib/fontinfo/*	fixes for other fonts
/usr/lib/vfont	directory containing fonts

SEE ALSO

troff(1), vfont(5), vpr(1)

BUGS

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Versatec fonts have different widths for individual characters than the fonts found on the typesetter, the following dodge was necessary: If you don't use the ".fp" troff directive then you get the widths of the standard typesetter fonts suitable for shipping the output of troff over the network to the computer center A machine for phototypesetting. If, however, you remount the R, I, B and S fonts, then you get the width tables for the Versatec.

NAME

vwidth – make troff width table for a font

SYNOPSIS

/usr/lib/vwidth *fontfile pointsize* **> ft*xx*.c**
cc -c ft*xx*.c
mv ft*xx*.o /usr/lib/font/ft*xx*

DESCRIPTION

Vwidth translates from the width information stored in the vfont style format to the format expected by troff. Troff wants an object file in a.out(5) format. (This fact does not seem to be documented anywhere.) Troff should look directly in the font file but it doesn't.

Vwidth should be used after editing a font with *fed(1)*. It is not necessary to use vwidth unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use vwidth if the physical width of the glyph (e.g. the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and vwidth run.

Vwidth produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the .o file) saved. The resulting file should be placed in /usr/lib/font in the file ft*xx* where is a one or two letter code that is the logical (internal to troff) font name. This name can be found by looking in the file /usr/lib/fontinfo/*fname** where *fname* is the external name of the font.

SEE ALSO

fed(1), vfont(5), troff(1), vtroff(1)

BUGS

Produces the C file using obsolete syntax that the portable C compiler complains about.

NAME

w — who is on and what they are doing

SYNOPSIS

w [**−h**] [**−s**] [user]

DESCRIPTION

W displays a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields displayed are: the users login name, the name of the tty the user is on, the time of day the user logged on (in hours:minutes), the idle time — that is, the number of minutes since the user last typed anything (in hours:minutes), the CPU time used by all processes and their children on that terminal (in minutes:seconds), the CPU time used by the currently active processes (in minutes:seconds), the name and arguments of the current process.

If a *user* name is included, output is restricted to that user.

OPTIONS

−h Suppress the heading.

−s Produce a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands.

−l Produce a long form of output, which is the default.

EXAMPLE

```
angel% w
  7:36am  up 6 days, 16:45,  1 users,  load average: 0.20, 0.23, 0.18
User     tty      login@  idle     JCPU   PCPU   what
henry    console  7:10am     1     10:05  4:31   w
angel%
```

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), ps(1), utmp(5)

BUGS

The notion of the 'current process' is muddy. The current algorithm is 'the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal'. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. In cases where no process can be found, *w* prints '—'.

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is 'charged' with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detecting background jobs. It will sometimes find a background job instead of the right one.

NAME

wait – await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with **&** or **bg** 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), csh(1)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for. (This bug does not apply to *csh*(1).)

NAME

wall – write to all users

SYNOPSIS

wall [file]

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

NAME

wc – word count

SYNOPSIS

wc [**–lwc**] [file **. . .**]

DESCRIPTION

Wc counts lines, words, and characters in the named *files*, or in the standard input if no *file* names appear. A word is a string of characters delimited by spaces, tabs, or newlines.

OPTIONS

If an argument beginning with one of 'lwc' is present, the specified counts are selected by the letters:

l Count lines.

w Count words.

c Count characters.

The default is **–lwc** (count lines, words, and characters).

EXAMPLE

```
angel% wc /usr/man/man1/{csh.1,sh.1,telnet.1}
   1876   11223   65895   /usr/man/man1/csh.1
    674    3310   20338   /usr/man/man1/sh.1
    260    1110    6834   /usr/man/man1/telnet.1
   2810   15643   93067   total
angel%
```

NAME

what – identify the version of files

SYNOPSIS

what files

DESCRIPTION

What searches the given *files* for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is **@(#)** at this printing) and prints out what follows until the first **"**, **>**, new-line, \, or null character. For example, if the C program in file ***program.c*** contains

```
char ident[] = "@(#)identification information";
```

and ***program.c*** is compiled to yield ***program.o*** and ***a.out,*** the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the SCCS command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

sccs(1), get(1), help(1), file(1).
The Source Code Control System in *Programming Tools for the Sun System*

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

It's possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

whatis — describe what a command is

SYNOPSIS

whatis command **...**

DESCRIPTION

Whatis looks up a given *command* and displays the header line from the manual section. You can then run the *man*(1) command to get more information. If the line starts 'name(section) **...** you can do **man section name** to get the documentation for it. Try **whatis ed** and then you should do **man 1 ed** to get the manual page for *ed.*

Whatis is actually just the **—f** option to the *man*(1) command.

FILES

/usr/man/whatis	Data base

SEE ALSO

man(1), catman(8)

NAME

whereis – locate source, binary, and/or manual for program

SYNOPSIS

whereis [**–sbm**] [**–u**] [**–BMS** dir **...** **–f**] *filename* **...**

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form **.***ext,* for example, **.**c. Prefixes of *s***.** resulting from use of source code control are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places.

OPTIONS

–b Search only for binaries.

–s Search only for sources.

–m Search only for manual sections.

–u Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus **whereis –m –u *** asks for those files in the current directory which have no documentation.

–B Change or otherwise limit the places where *whereis* searches for binaries.

–M Change or otherwise limit the places where *whereis* searches for manual sections.

–S Change or otherwise limit the places where *whereis* searches for sources.

–f Terminates the last directory list and signals the start of file names, and *must* be used when any of the **–B**, **–M**, or **–S** options are used.

EXAMPLE

Find all files in */usr/bin* which are not documented in */usr/man/man1* with source in */usr/src/cmd*:

```
angel% cd /usr/ucb
angel% whereis –u –M /usr/man/man1 –S /usr/src/cmd –f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

BUGS

Since *whereis* uses *chdir*(2) to run faster, pathnames given with the **–M**, **–S**, or **–B** must be full; that is, they must begin with a '/'.

NAME

which – locate a program file including aliases and paths (*csh* only)

SYNOPSIS

which [name] ...

DESCRIPTION

Which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's .cshrc file.

FILES

~/.cshrc	source of aliases and path values

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

BUGS

Only aliases and paths from ~/.cshrc are used; importing from the current environment is not attempted. Must be executed by a csh, since only csh's know about aliases.

To compensate for *~/.cshrc* files in which aliases depend upon the **prompt** variable being set, *which* sets this variable. If the *~/.cshrc* produces output or prompts for input when **prompt** is set, *which* may produce some strange results.

NAME

who – who is on the system

SYNOPSIS

who [who-file] [**am i**]

DESCRIPTION

Used without arguments, *who* lists the login name, terminal name, and login time for each current UNIX user. *Who* gets this information from the */etc/utmp* file.

If a filename argument is given, the named file is examined instead of */etc/utmp*. Typically the named file is */usr/adm/wtmp*, which contains a record of all logins since it was created. In this case, *who* lists logins, logouts, and crashes. Each login is listed with user name, terminal name (with '*/dev/*' suppressed), and date and time. Logouts produce a similar line without a user name. Reboots produce a line with '~' in place of the device name, and a fossil time indicating when the system went down. Finally, the adjacent pair of entries '|' and '}' indicate the system-maintained time just before and after a *date* command changed the system's idea of the time.

With two arguments, as in 'who am i' (and also 'who are you'), *who* tells who you are logged in as: it displays your hostname, login name, terminal name, and login time.

EXAMPLES

```
angel% who am i
angel!henry      ttyp0   Apr 27 11:24
angel%

krypton% who
mktg      ttym0 Apr 27 11:11
shannon             ttyp0  Apr 27 11:25
henry     ttyp1  Apr 27 11:30
krypton%
```

FILES

/etc/utmp

SEE ALSO

whoami(1), getuid(2), w(1)

NAME

whoami – display effective current username

SYNOPSIS

whoami

DESCRIPTION

Whoami displays the username of whoever is currently logged in. *Whoami* works even if you are logged in as the super-user, while 'who am i' does not since it gets its information from the */etc/utmp* file.

FILES

/etc/passwd	username data base

SEE ALSO

who(1)

NAME

whois – DARPA Internet user name directory service

SYNOPSIS

whois [–h host] identifier

DESCRIPTION

whois searches for an ARPAnet directory entry for an *identifier* which is either a name (such as "Smith") or a handle (such as "SRI-NIC"). You can force a name-only search by preceeding the name with a period; you can force a handle-only search by preceeding the handle with an exclamation point. For example, typing:

whois Smith	looks for name or handle SMITH.
whois !SRI-NIC	looks for handle SRI-NIC only.
whois .Smith, John	looks for name JOHN SMITH only.

Adding "..." to the name or handle argument will match anything from that point; that is, "ZU..." will match ZUL, ZUM, etc.

If you are searching for a group or organization entry, you can have the entire membership list of the group displayed with the record by preceeding the argument with an astersik ("*").

You may of course use an exclamation point and asterisk, or a period and asterisk together.

NAME

write – write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from your standard input to *user*'s screen.

When you type a *write* command, the person you're writing to sees a message like this:

```
Message from hostname!yourname on yourttyname at hh:mm . . .
```

After typing the *write* command, enter the text of your message. What you type appears line-by-line on the other user's screen. Conclude by typing an end of file indication (^D) or an interrupt. At this point *write* displays 'EOT' on your recipient's screen and exits.

To write to a user who is logged in more than once, use the *ttyname* argument to indicate the appropriate terminal name.

You can grant or deny other users permission to write to you by using the *mesg* command (default allows writing). Certain commands, *nroff* and *pr*(1) in particular, don't allow anyone to write to you while you are using them in order to prevent messy output.

If *write* finds the character '!' at the beginning of a line, it calls the shell to execute the rest of the line as a command.

Two people can carry on a conversation by *write*'ing to each other. When the other person receives the message indicating you are writing to him, he can then *write* back to you if he wishes. However, since you are now simultaneously typing and receiving messages, you end up with garbage on your screen unless you work out some sort of scheduling scheme with your partner. You might try the following conventional protocol: when you first write to another user, wait for him to write back before starting to send. Each person should end each message with a distinctive signal — **-o-** (for 'over') is standard — so that the other knows when to begin a reply. To end your conversation, type **-oo-** (for 'over and out') before finishing the conversation.

EXAMPLE

Here is an example of a short dialog between two people on different terminals. Two users called Horace and Eudora are logged in on a system called jones. To illustrate the process, both users' screens are shown side-by-side:

```
         Eudora's Terminal                          Horace's Terminal
                                            Horace is staring at his screen
jones% write horace                         Message from jones!eudora on tty09 at 17:05 ...
how about a squash game tonight? -o-        how about a squash game tonight? -o-
                                            jones% write eudora
                                            I'm playing tiddlywinks with Carmeline -o-
Message from jones!horace on tty03 at 17:06 ...
I'm playing tiddlywinks with Carmeline -o-
How about the beach on Sunday? -o-          How about the beach on Sunday? -o-
                                            Sorry, I'm washing my tent that day -o-
Sorry, I'm washing my tent that day -o-
See you when I get back from Peru -oo-      See you when I get back from Peru -oo-
^D
jones%                                      EOF
                                            I hear rack of llama is very tasty -oo-
                                            ^D
I hear rack of llama is very tasty -oo-
EOF                                         jones%
```

FILES

/etc/utmp	to find user
/bin/sh	to execute ‘!’

SEE ALSO

mesg(1), who(1), mail(1), talk(1)

NAME

xsend, xget, enroll – secret mail

SYNOPSIS

xsend *username*
xget
enroll

DESCRIPTION

These commands implement a secure communication channel; the channel is like *mail*(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use *enroll*; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use *xget*. It asks for your password, then gives you the messages.

To send secret mail, use *xsend* in the same manner as the ordinary mail command. Unlike *mail*, *xsend* accepts only one target. A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

/usr/spool/secretmail/*.key	keys
/usr/spool/secretmail/*.[0-9]	messages

SEE ALSO

mail (1)

BUGS

The knapsack public-key cryptosystem is known to be breakable.

RESTRICTIONS

These facilities are not available on software shipped outside the U.S.

NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**–v**] [**–c**] [**–**] [file]

DESCRIPTION

Xstr maintains a file called *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

```
xstr –c name
```

extracts the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c,* to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

```
xstr
```

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

```
xstr name
```

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '–' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc –E name.c | xstr –c –
cc –c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

OPTIONS

–c *file* Take C source text from *file*.

–v Verbose: display a progress report indicating where new or duplicate strings were found.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' doesn't touch *strings*

SEE ALSO

mkstr(1)

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc — yet another compiler-compiler

SYNOPSIS

yacc [**−vd**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a function named *yyparse*. The *yyparse* function must be loaded with the lexical analyzer *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*-produced parsers.

OPTIONS

−v Prepare the file *y.output* containing a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

−d Generate the file *y.tab.h* with the *define* statements that associate the *yacc*-assigned 'token codes' with the user-declared 'token names' so that source files other than *y.tab.c* can access the token codes.

FILES

y.output	description of parsing tables and conflict report
y.tab.c	output parser
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974
Yacc — Yet Another Compiler Compiler in *Programming Tools for the Sun Workstation*

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process should be active in a given directory at a time.

NAME

yes – be repetitively affirmative

SYNOPSIS

yes [expletive]

DESCRIPTION

Yes repeatedly outputs "y", or if *expletive* is given, that is output repeatedly. Termination is by typing an interrupt character.

NAME

ypcat - print values in a YP data base

SYNOPSIS

ypcat [**−a**] [**−k**] [**−t**] [**−d** domainname] keyword

DESCRIPTION

Ypcat prints out values in a yellow pages (YP) map specified by a *keyword*, which may be either a *mapname* or an abbreviated mapname. Since *ypcat* employs YP network services, the location of the YP server cannot be supplied.

Suppose you want to look at the system password file. Each machine has a file */etc/passwd*, containing only a small amount of information. The network-wide password database is served by the YP, and holds information in a map named *passwd.byname*, whose abbreviated name is *passwd*. Thus, to see the system passwords, you type:

```
ypcat passwd
```

Here is a list of abbreviated mapnames, full mapnames, and corresponding system files:

passwd	passwd.byname	/etc/passwd
group	group.byname	/etc/group
hosts	hosts.byaddr	/etc/hosts
networks	networks.byaddr	/etc/networks
services	services.byname	/etc/services
protocols	protocols.bynumber	/etc/protocols
netgroup	netgroup	/etc/netgroup

A list of all maps in the default domain can be seen by typing:

```
ypcat -k ypmaps
```

A list of all the known domains can be seen by typing:

```
ypcat -d yp_private -k ypdomains
```

OPTIONS

−a Get all the key_value pairs back from the YP, including private symbols. These are uninteresting to almost everyone.

−k Dump the keys, as well as the values. Keys are already within the values for maps interesting to most users. Here again, the extra information you get back is generally uninteresting. Exceptions are those maps in which the value is null (that is, all of the information in the map is in the keys), or maps in which the key is not part of the value. None of the maps derived from files that have an ASCII version in */etc* fall into those classes.

−t Inhibit translation of abbreviated mapname to mapname. For example, *ypcat −t passwd* will fail because there is no file named *passwd*, while *ypcat passwd* will be translated to *ypcat passwd.byname*. The purpose of this option is to let you specify a mapname which is also a keyword name.

−d Used to specify a domain other that the default domain. The default domain is returned by *domainname*(8). The default domain is the one that is interesting to most users.

SEE ALSO

ypfiles(8), yppush(8), newpasswd(8)

NAME

yppasswd – change login password in yellow pages

SYNOPSIS

yppasswd [name]

DESCRIPTION

This command changes (or installs) a password associated with the user *name* (your own name by default) in the yellow pages. The yellow pages password may be different from the one on your own machine.

Yppasswd prompts for the old yellow pages password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

SEE ALSO

passwd(1), ypfiles(5), yppasswdd(8C)

BUGS

The update protocol passes all the information to the server in one rpc call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

NAME

adventure – an exploration game

SYNOPSIS

/usr/games/adventure

DESCRIPTION

The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type ‘quit’; to save a game for later resumption, type ‘suspend’.

BUGS

Saving a game creates a large executable file instead of just the information needed to resume the game.

NAME

arithmetic – provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [**+–x/**] [range]

DESCRIPTION

Arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+–x/** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is **+–**

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon – the game of backgammon

SYNOPSIS

backgammon [–] [n r w b pr pw pb t*term* s*file*]

DESCRIPTION

Backgammon lets you play backgammon against the computer or against a 'friend'. All commands only are one letter, so you don't need to type a carriage return, except at the end of a move. *Backgammon* is mostly self documenting, so that a question mark (?) will usually get some help. If you answer 'y' when *backgammon* asks if you want the rules, you will get text explaining the rules of the game, some hints on strategy, instruction on how to use *backgammon,* and a tutorial consisting of a practice game against the computer. A description of how to use *backgammon* can be obtained by answering 'y' when it asks if you want instructions. The possible arguments for backgammon (most are unnecesary but some are very convenient) consist of:

n don't ask for rules or instructions

r player is red (implies n)

w player is white (implies n)

b two players, red and white (implies n)

pr print the board before red's turn

pw print the board before white's turn

pb print the board before both player's turn

t*term* terminal is type *term*, uses */etc/termcap*, otherwise uses the TERM environment variable.

s*file* recover previously saved game from *file*. This can also be done by executing the saved file, that is, typing its name in as a command.

Arguments may be optionally preceded by a – sign. Several arguments may be concatenated together, but not after 's' or 't' arguments, since they can be followed by an arbitrary string. Any unrecognized arguments are ignored. An argument of a lone – gets a description of possible arguments.

If *term* has capabilities for direct cursor movement. *backgammon* 'fixes' the board after each move, so the board does not need to be reprinted, unless the screen suffers some horrendous malady. Also, any 'p' option will be ignored.

QUICK REFERENCE

When *backgammon* prompts by typing only your color, type a space or carriage return to roll, or

d to double

p to print the board

q to quit

s to save the game for later

When *backgammon* prompts with 'Move:', type

p to print the board

q to quit

s to save the game

or a *move,* which is a sequence of

s-f move from **s** to **f**

s/r move one man on **s** the roll **r** separated by commas or spaces and ending with a newline. Available abbreviations are

s-f1-f2 means **s-f1,f1-f2**

s/r1r2 means **s/r1,s/r2**

Use 'b' for bar and 'h' for home, or 0 or 25 as appropriate.

FILES

/usr/games/teachgammon rules and tutorial
/etc/termcap terminal capabilities

BUGS

Backgammon's strategy needs much work.

NAME

banner − print large banner on printer

SYNOPSIS

/usr/games/banner [**−w***n*] message ...

DESCRIPTION

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

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

BUGS

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

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

NAME

bcd – convert to antique media

SYNOPSIS

/usr/games/bcd text

DESCRIPTION

Bcd converts the literal *text* into a form familiar to old-timers.

SEE ALSO

dd(1)

NAME

bdemos – demonstrate Sun Monochrome Bitmap Display

SYNOPSIS

/usr/demo/bballs
/usr/demo/bbounce
/usr/demo/bdemos
/usr/demo/bjump
/usr/demo/bphoto *file*
/usr/demo/brotcube

DESCRIPTION

Bdemos is a collection of simple demonstration programs for the Sun Monochrome Bitmap Display. Each program is briefly described below. Unless otherwise noted, each program should be terminated by typing the appropriate key (usually DELETE or ^C) to generate an interrupt signal.

bballs colliding balls demo

bbounce bouncing square demo

bdemos a collection of demos

This program has a menu for selection of several different demos. After typing a key to select a particular demo, the user may type ^C to get back the menu. Type 'q' to quit.

bjump simulated jump to hyperspace

bphoto *file*

dither monochrome image *file* to bitmap display

Image files suitable for display by this program are in */usr/demo/bwpix*.

brotcube black and white spinning cube

FILES

/usr/demo/bwpix

SEE ALSO

draw(6)

NAME

boggle – play the game of boggle

SYNOPSIS

/usr/games/boggle [+] [++]

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

bsuncube – view 3-D Sun logo

SYNOPSIS

/usr/demo/bsuncube

DESCRIPTION

Bsuncube allows the user to view a cube from various positions with hidden faces removed. The faces of the cube consist of the Sun logo. The viewing position is selected using the mouse. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display.

The program operates in two modes: DisplayObject mode and SelectView mode. The program starts in DisplayObject mode.

DisplayObject: The cube is displayed in 3-D perspective with hidden faces removed. Type "q" while in this mode to exit the program. Hit mouse button 3 to switch to SelectView mode.

SelectView: Schematic projections of the outline of the cube are shown and the mouse is used to select a viewing position. Use button 1 to set x and button 2 to set y in the *Front View*. Use button 2 to set z in the *Top View*. Hit button 3 to switch to DisplayObject mode.

The view shown in DisplayObject mode is drawn using the conventions that the viewer is always looking from the viewing position toward the center of the cube and that the positive y axis on the screen is the projection of the positive y axis in 3-D cube coordinates.

NAME

canfield, cfscores – the solitaire card game canfield

SYNOPSIS

/usr/games/canfield
/usr/games/cfscores

DESCRIPTION

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase ones chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (the initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

FILES

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

NAME

chase – Try to escape to killer robots

SYNOPSIS

/usr/games/chase [*nrobots*] [*nfences*]

DESCRIPTION

The object of the game chase is to move around inside of the box on the screen without getting eaten by the robots chasing and without running into anything.

If a robot runs into another robot while chasing you, they crash and leave a junk heap. If a robot runs into a fence, it is destroyed.

If you can survive until all the robots are destroyed, you have won!

If you do not specify either *nrobots* or *nfences,* chase will prompt you for them.

NAME

chess – the game of chess

SYNOPSIS

/usr/games/chess

DESCRIPTION

Chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

DIAGNOSTICS

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

FILES

/usr/games/lib/chess.book	book of opening moves

BUGS

Pawns may be promoted only to queens.

NAME

chesstool — window-based front-end to chess program

SYNOPSIS

chesstool [*chess_program*]

DESCRIPTION

Chesstool is a window-based front-end to the *chess*(6) program. Used without options, *chesstool* uses */usr/games/chess*; you can designate any alternate program which uses the same command syntax as *chess*(6) with the *chess_program* argument.

When *chesstool* starts up, it displays a large window with three subwindows. The first subwindow displays messages — "Illegal move", for example. The second subwindow is an options subwindow; options are described below. The final subwindow is a chessboard display with white and black pieces and two (advisory only) timekeeping clocks.

Make your moves with the mouse: select a piece by positioning the arrow cursor over the piece and pressing the left mouse button down, then drag the piece to the destination square, and release the button. The cursor will then turn to an hourglass icon while the system plays.

Options in the options subwindow may be selected with either the left or middle mouse buttons. These options are:

(Last Play) Show the last play made.

(Undo) Undo your last move and the machine's response.
Once the game is over, it is not possible to restart it, so undo will update the board, but the game cannot be continued from that position.

[Flash] Flash when the machine has completed its move.
In flash mode, if the *chesstool* is open, the piece moved by the system on its play will flash until you make your move. If the *chesstool* is iconic, the entire icon will flash when the machine has made its move. Thus you can 'Close' the *chesstool* and be alerted when it's your turn to move. To turn flash mode off, select flash again.

(Machine White) Start a new game with the machine playing white.

(Human White) Start a new game with the machine playing black.

There are two moves which are special: castling and capturing a pawn *en passant.* To castle, move the king only. The position of the rook will automatically be updated. Since the king moves two squares when castling, the move is unambiguous. To capture *en passant*, move the pawn to the square occupied by the opposing pawn which will be captured.

SEE ALSO

chess(6)

NAME

ching – the book of changes and other cookies

SYNOPSIS

/usr/games/ching [hexagram]

DESCRIPTION

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams,* each symbolized by a particular arrangement of six straight (——) and broken (– –) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each hexagram consists of two major sections. The **Judgement** relates specifically to the matter at hand (E.g., "It furthers one to have somewhere to go.") while the **Image** describes the general attributes of the hexagram and how they apply to one's own life ("Thus the superior man makes himself strong and untiring.").

When any of the lines has the value six or nine, it is a moving line; for any such line there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second hexagram (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow–stalks or tossed coins. The resulting hexagram will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, the UNIX *oracle* simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process id and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin–toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try *fortune*(6).

SEE ALSO

It furthers one to see the great man.

DIAGNOSTICS

The great prince issues commands,
Founds states, vests families with fiefs.
Inferior people should not be employed.

BUGS

Waiting in the mud
Brings about the arrival of the enemy.

If one is not extremely careful,
Somebody may come up from behind and strike him.
Misfortune.

NAME

colordemos – demonstrate Sun Color Graphics Display

SYNOPSIS

/usr/demo/cballs
/usr/demo/cdraw
/usr/demo/cphoto *file*
/usr/demo/cpipes
/usr/demo/cshowmap *file*
/usr/demo/csnow
/usr/demo/csuncube
/usr/demo/csunlogo
/usr/demo/cvlsi

DESCRIPTION

Colordemos is a collection of simple demonstration programs for the Sun Color Graphics Display. Each program is briefly described below. To exit each program, send an interrupt signal by typing the appropriate key (usually CONTROL C).

cballs colliding balls on color display

cdraw draw on the color display; see *draw*(6) for an explanation of how to use *cdraw*.

cphoto *file*
display dithered color file on color display. Files suitable for display are in */usr/demo/colorpix*.

cpipes colliding pipes on color display

cshowmap *file*
display maps. Files suitable for display are in */usr/demo/segments*.

csnow color kaleidoscope

csuncube multicolored Sun logo

csunlogo shaded Sun logo

cvlsi color vlsi layout demo

FILES

/usr/demo/colorpix
/usr/demo/segments

NAME

cribbage — the card game cribbage

SYNOPSIS

/usr/games/cribbage [**−req**] *name* ...

DESCRIPTION

Cribbage plays the card game cribbage, with *cribbage* playing one hand and the user the other. *Cribbage* initially asks the user if the rules of the game are needed — if so, *cribbage* displays the appropriate section from *According to Hoyle* with *more*(1).

OPTIONS

−e Provide an explanation of the correct score when the player makes mistakes scoring his hand or crib. This is especially useful for beginning players.

−q Print a shorter form of all messages — this is only recommended for users who have played the game without specifying this option.

−r Instead of asking the player to cut the deck, *cribbage* will randomly cut the deck.

PLAYING CRIBBAGE

Cribbage first asks the player whether he wishes to play a short game ("once around", to 61) or a long game ("twice around", to 121). A response of 's' results in a short game, any other response plays a long game.

At the start of the first game, *cribbage* asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, *cribbage* first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, *cribbage* cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the later case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. *Cribbage* keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. *Cribbage* requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

SPECIFYING CARDS

Cards are specified as *rank* followed by *suit*. The *rank*s may be specified as one of **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **t**, **j**, **q**, and **k**, or alternatively, one of **ace**, **two**, **three**, **four**, **five**, **six**, **seven**, **eight**, **nine**, **ten**, **jack**, **queen**, and **king**. *Suit*s may be specified as **s**, **h**, **d**, and **c**, or alternatively as **spades**, **hearts**, **diamonds**, and **clubs**. A card may be specified as *rank suit*, or *rank* of *suit*. If the single letter *rank* and *suit* designations are used, the space separating the *suit* and *rank* may be left out. Also, if only one card of the desired *rank* is playable, typing the *rank* is sufficient. For example, if your hand was **2h**, **4d**, **5c**, **6h**, **jc**, **kd** and you wanted to discard the king of diamonds, you could type any of **k**, **king**, **kd**, **k d**, **k of d**, **king d**, **king of d**, **k diamonds**, **k of diamonds**, **king diamonds**, or **king of diamonds**.

FILES

/usr/games/cribbage

NAME

draw – interactive graphics drawing

SYNOPSIS

/usr/demo/bdraw
/usr/demo/cdraw

DESCRIPTION

The *draw* programs are menu-driven programs which use the mouse, keyboard, bitmap display and optionally the color display to draw objects, drag them around, save them on disk, and so on. *Bdraw* is the draw program for the black and white display and *cdraw* is the program for driving the color display.

The main menu items are selected by moving the mouse cursor and pressing the left mouse button. To redraw the display, point at the left edge of the main menu box and press the left button. The main menu items are:

New Seg xlate
Open a new translatable segment. A segment is a collection of attributes and primitives (lines, text, polygons, etc.). A translatable segment may subsequently be positioned.

New Seg xform
Open a new transformable segment. A transformable segment may subsequently be rotated, scaled, or positioned.

Delete Seg
To delete a segment, point at any primitive in the segment and press the left button.

Lines To add line primitives to the currently open segment, position cursor, press the left button, ... press right button to quit.

Polygon To add a polygon primitive to the currently open segment, position the cursor, press the left button, ... press the right button to terminate the boundary definition. Polygons are filled with the current fill attribute.

Raster To add a raster primitive to the currently open segment, position the cursor, press the left button to reposition the box, adjust the box by moving the mouse, press the right button to create the raster primitive comprising the boxed bitmap. A 'rasterfile' is also created on disk for hardcopy purposes (see */usr/include/rasterfile.h*). This 'rasterfile' file may be spooled to a Versatec printer/plotter for hardcopy after exiting from the draw program. The command to do this is **lpr –v rasterfile**.

Text To add a text primitive to the currently open segment, position cursor, press left button, type the text string at the keyboard (back space works), hit return. Text is drawn with the current text attributes.

Marker To add marker primitives to the currently open segment, position cursor, press the left button to place marker, ... press the right button to quit.

Position To position a segment, point at any primitive in the segment, press left button, position the segment, press right button to quit.

Rotate To rotate a transformable segment, point at any primitive in the segment, press left button, move mouse to rotate, press right button to quit.

Scale To scale a transformable segment, point at any primitive in the segment, press the left button, move mouse to scale in x or y, press right button to quit.

Attributes
This item brings up the attribute menu. To select an attribute such as text font, region fill texture (color), linestyle, or line width, point at the item and press the left

button. Point at the left edge of the menu box to quit.

Save Seg To save a segment on a disk file, point at the segment, press the left button, type the disk file name, hit return.

Restore Seg
To restore a previously saved segment from disk, type file name, hit return.

Exit Exit the draw program.

BUGS

Rasters and raster text do not scale or rotate. If segments completely overlap, only the last one drawn may be picked by pointing with the mouse. This also applies to the menu segments! Therefore, don't cover them up with polygons. If aborted with your interrupt character, you must give the 'reset' command to turn keyboard echo back on and to reset -cbreak. Therefore, use the Exit item in the main menu to exit the program.

NAME

fish – play "Go Fish"

SYNOPSIS

/usr/games/fish

DESCRIPTION

Fish plays the game of "Go Fish", a children's card game. The object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; the default is pretty dumb.

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [–] [**–wsla**]

DESCRIPTION

Fortune with no arguments prints out a random adage. The flags mean:

–w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

–s Short messages only.

–l Long messages only.

–a Choose from either list of adages.

FILES

/usr/games/lib/fortunes.dat

NAME

gammontool – play a game of backgammon

SYNOPSIS

gammontool [path]

DESCRIPTION

Gammontool paints a backgammon board on the screen, and then lets you play against the computer. It must be run in SunWindows. The optional *path* argument specifies an alternate move-generating program, which must be specially designed to run with *gammontool*.

The game has three subwindows: an option window on top, a message window in the middle, and a large board on the bottom. The buttons in the option window are used to restart, double, etc. The message window has two lines: the first tells whose turn it is, and the second displays any errors that occur.

The initial roll. To start the game, roll the dice to determine who goes first. Move the mouse arrow onto the board and click the left button. One die appears on each side of the board: the die on the left is yours, and the die on the right is the computer's. If your roll is greater, then you move; if not, the computer makes a move.

Making your move. When it is your turn, "Your move" appears in the message window. Place the mouse over any piece of your color, and click the left button. While holding down the button, move the mouse to drag the piece; the piece follows the mouse until you release the button. The tool checks each move and does not allow illegal moves. When you have made as many moves as you can, the computer takes its turn; after it finishes, you may either roll again, or double.

Doubling. To double, click the *Double* button in the option window and wait for the computer's response. If the computer doubles you, a message is displayed and you must answer with the *Accept Double* or *Refuse Double* buttons. The *Forfeit* button can also be used to refuse a double. If the game is doubled, a doubling cube with the proper value is displayed on the bar strip. If the number is facing up, then you may double next. If the number is upside down, it is the computer's turn to double.

Other buttons. If you want to change your move before you have finished it, use the *Redo Move* or *Redo Entire Move* buttons in the option window. *Redo Entire Move* replaces all of the pieces you have moved so that you can redo them all. *Redo Move* only replaces the last piece you moved, so it is useful when you roll doubles and want to redo only the last piece you moved. Note that once you have made all of the moves your roll permits, play passes immediately to the computer, so you cannot redo the very last move. The *Show Last Move* button allows you to see the last move again.

Leaving the game. If you want to quit playing backgammon, use the *Quit* button. If you want to forfeit the game, use the *Forfeit* button. The computer penalizes you by taking a certain number of points, but the program does not terminate.

To play another game after winning, losing, or forfeiting, click either the *Human is white* or *Human is black* buttons, depending on which color you want. Your pieces always move from the top right to the bottom right of the board, regardless of your color. As an additional cue as to your color, your dice is always displayed on the left half of the board.

Log file. If a there is a *gammonlog* file your home directory, *gammontool* keeps a log of the games played. Each move and double gets recorded, along with the winners and accumulated scores.

FILES

~/gammonlog	log of games played

BUGS

The default strategy used by the computer is very poor.

If a move uses more than one die (for instance if you roll 5, 6 and move 11 spaces without landing in the middle) it is unpredictable where the piece will touch down at in the middle. The program will always make the move if possible, but if two midpoints would work and there is a blot on one of them, it is much better to explicitly hit the blot and then move the piece the rest of the way.

NAME

hangman — Computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In *hangman,* the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words	On-line word list

NAME

mille – play Mille Bornes

SYNOPSIS

/usr/games/mille [file]

DESCRIPTION

Mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bornes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P Pick a card from the deck. This card is placed in the 'P' slot in your hand.

D Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a carriage-return or space. The carriage-return or space is required to allow recovery from typos which can be very expensive, like discarding safeties.

U Use a card. The card is again indicated by its number, followed by a carriage-return or space.

O Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.

Q Quit the game. This will ask for confirmation, just to be sure. Hitting <DELETE> (or <RUBOUT>) is equivalent.

S Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a carriage-return without a name, the save will be terminated and the game resumed.

R Redraw the screen from scratch. The command ^L (control 'L') will also work.

W Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "**S**" command to reattempt the save. (The game itself is a product of Parker Brothers, Inc.)

SEE ALSO

curses(3X)

CARDS

Here is some useful information. The number in brackets after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas [2]	Gasoline [6]	Extra Tank [1]
Flat Tire [2]	Spare Tire [6]	Puncture Proof [1]
Accident [2]	Repairs [6]	Driving Ace [1]
Stop [4]	Go [14]	Right of Way [1]
Speed Limit [3]	End of Limit [6]	

25 – [10], 50 – [10], 75 – [10], 100 – [12], 200 – [4]

RULES

Object: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. With the exception of the *speed limit* card, they can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA** (unlabeled): This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaches 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastyness.

Go (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

Stop is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

Speed Limit is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

End of Limit is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

Out of Gas is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

Flat Tire is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.

Accident is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn*.

Right of Way prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.

Extra Tank When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

Puncture Proof When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

Driving Ace When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand*. A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action*.

Coup Fouré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bornes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.

Each Safety: 100 points for each safety in the Safety area.

All 4 Safeties: 300 points if all four safeties are played.

Each Coup Fouré: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.

Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.

Delayed Action: 300 points bonus for finishing after the deck was exhausted.

Extension: 200 points bonus for completing a 1000 mile trip.

Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop – Monopoly game

SYNOPSIS

/usr/games/monop [file]

DESCRIPTION

Monop is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", *i.e.*, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, e.g., a name, place or person, you can type '?' to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

A Summary of Commands:

quit: quit game: This allows you to quit the game. It asks you if you're sure.

print: print board: This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

Name	The first ten characters of the name of the square
Own	The *number* of the owner of the property.
Price	The cost of the property (if any)
Mg	This field has a '*' in it if the property is mortgaged
#	If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
Rent	Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings:
List your own holdings, *i.e.*, money, get-out-of-jail-free cards, and property.

holdings: holdings list: Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type "done".

shell: shell escape: Escape to a shell. When the shell dies, the program continues where you left off.

mortgage:
mortgage property: Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage:
unmortgage property: Unmortgage mortgaged property.

buy: buy houses: Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: sell houses: Sets up a list of monopolies from which you can sell houses. it operates in an analogous manner to *buy*

card: card for jail: Use a get-out-of-jail-free card to get out of jail. If you're not in jail, or you don't have one, it tells you so.

pay: pay for jail: Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you're not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: save game: Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the *monop* command, or by using the *restore* command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore: restore game: Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the <RETURN> key instead of a command, it is the same as typing *roll*.

FILES

/usr/games/lib/cards.pck Chance and Community Chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

number – convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

Number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

quiz — test your knowledge

SYNOPSIS

/usr/games/quiz [**−i** file] [**−t**] [category1 category2]

DESCRIPTION

Quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, *quiz* gives instructions and lists the available categories.

Quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **−t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **−i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category ':' line
category  = alternate | category '¦' alternate
alternate = empty | alternate primary
primary   = character | '[' category ']' | option
option    = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with *sh*(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a¦ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain — animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

Rain's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use *termcap*, the TERM environment variable must be set (and exported) to the type of the terminal being used.

FILES

/etc/termcap

NAME

sail – multi-user wooden ships and iron men

SYNOPSIS

sail [**−x**] [**num**]

DESCRIPTION

Sail is a computer version of Avalon Hill's game of fighting sail originally developed by S. Craig Taylor.

NOTES

Sail is really two programs in one. Each player keeps track of his own ship plus a *DRIVER* program is execl'd (by the first player) to keep track of the computer's ships.

The player is given the first available ship in a scenario and the computer takes the rest. Obviously the more ships in your game, the longer the *DRIVER* will take to move them. If additional players join the game, they will be given ships and the *DRIVER* will have less work to do.

HISTORICAL INFO

Old Square Rigger's were very maneuverable ships capable of intricate sailing. Their one main disadvantage was being unable to sail very close to the wind. The design of wooden ship allowed only for the guns to bear to the left and right sides. A few guns of small aspect (usually 6 or 9 pounders) could point forward, but their effect would be small compared to a 68 gun broadside of 24 pounders. The guns bear approximately like so:

```
   \
    b----------------
 ---0
    \
     \
      \    up to a range of ten (for round shot)
       \
        \
         \
```

An interesting phenomenon occurred when a broadside could fire down the length of an enemy ship. The shot tended to bounce along the deck and did several times more damage. This phenomenon was called a rake. It happened that a stern rake (firing from the stern to the bow) occasioned more damage than a bow rake, so that was the most desirable.

```
   \
   b---------------
---0
    \
    \      0a  ---  Stern rake!
     \
      \
       \
        \
         \
```

Most ships were equipped with Carronades which were very large, close range cannons. The carronades have a range of two in this game and can considerably add to your fire-power when they come to bear. If the distance to the target ship is greater than 6, the guns can only fire at the rigging. A ship's guns could fire a variety of ammunition. For example:

ROUND

Range of 10. Good for hull or rigging hits.

DOUBLE

Range of 1. Extra good for hull or rigging hits. Double takes two turns to load.

CHAIN

Range of 3. Excellent for tearing down rigging. Cannot damage hull or guns, though.

GRAPE

Range of 1. Devastating against enemy crews.

When a ship has been battered into a hulk (zero hull), it has no choice but to surrender to the firing ship. This ceremony is called ´striking your colours.´ A struck ship has a chance of exploding or sinking after a while. When a ship surrenders, its point value is given to the aggressor. When a ship is captured, twice the point value is awarded the victor.

Normally, ships sailed into battle with greatly shortened sail to avoid excessive damage to the precious rigging. However, in this game the player can increase to full sails and move much faster if he wishes. But, all rigging hits incurred with full sails set are doubled. The direction rose displayed on the sample screen gives the maximum speeds possible for a specific ship at all attitudes to the wind. The full sail speeds are in parenthesis.

Repairs can be made at the slow rate of two (hull, gun, or rigging) hits restored per three turns.

Ships of class 3 or greater drift when there is wind at the rate of one ´square´ per turn. Ships of the Line drift one ´square´ every other turn.

INSTRUCTIONS

Sail follows the Avalon Hill advanced rules very closely using the optional rules for 'exploding ships', 'full sails', and some others. A few unique commands have been added which seemed to be helpful on the reduced screen. 'i' is such a command.

Boarding had to go through a major revision. To prevent immediate capture of an unprepared crew (fouling is often not reported until after boarding has commenced) the boarded ship automatically fights defensively (at a small disadvantage) if no DBP's have been prepared.

The Order of Play has been eliminated for the player, but the *DRIVER* still abides by it.

The commands for the player were designed to be as intelligent as possible to save typing. Some of the nuances I developed should be explained.

~ Your prompt

The others I will illustrate with examples.

```
move(3, 2): rll        /* 3 movements max, of which two
                       may be 45' turns. */

move(3,'2): 1r1         /* 3 movements max of which two may
                       be 45' turns, but the ship must
                       move ahead before turning (there
                       is a loss of headway after
                       drifting) */

move(0,'0): r          /* You can always make one turn
                       even when you can't move straight
                       ahead. */
```

If you are grappled, fouled, or out of crew, you cannot move of course.

COMMANDS

'f' Fire broadsides if they bear
'l' Reload
'm' Move (see above & below)
'i' Ask lookout for closest ship
'I' Ask lookout for closest enemy ship
's' Send a message around the fleet
'b' Attempt to board an enemy ship
'L' Unload broadsides (to change ammo)
'B' Recall boarding parties
'c' Change set of sail
'r' Repair
'u' Attempt to unfoul
'g' Grapple/ungrapple
'^L' Redraw screen
'q' Quit

SCENARIOS

Ranger vs. Drake:

Wind from the N, blowing a fresh breeze.

(a) Ranger	19 gun Sloop (crack crew) (7 pts)
(b) Drake	17 gun Sloop (crack crew) (6 pts)

The Battle of Flamborough Head:

Wind from the S, blowing a fresh breeze.

This is John Paul Jones' first famous battle. Aboard the Bonhomme Richard, he was able to overcome the Serapis's greater firepower by quickly boarding her.

(a) Bonhomme Rich	42 gun Corvette (crack crew) (11 pts)
(b) Serapis	44 gun Frigate (crack crew) (12 pts)

Arbuthnot and Des Touches:

Wind from the N, blowing a gale.

(b) America	64 gun Ship of the Line (crack crew) (20 pts)
(b) Befford	74 gun Ship of the Line (crack crew) (26 pts)
(b) Adamant	50 gun Ship of the Line (crack crew) (17 pts)
(b) London	98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Royal Oak	74 gun Ship of the Line (crack crew) (26 pts)
(f) Neptune	74 gun Ship of the Line (average crew) (24 pts)
(f) Duc Bougogne	80 gun 3 Decker SOL (average crew) (27 pts)
(f) Conquerant	74 gun Ship of the Line (average crew) (24 pts)
(f) Provence	64 gun Ship of the Line (average crew) (18 pts)
(f) Romulus	44 gun Ship of the Line (average crew) (10 pts)

Suffren and Hughes:

Wind from the S, blowing a fresh breeze.

(b) Monmouth	74 gun Ship of the Line (average crew) (24 pts)
(b) Hero	74 gun Ship of the Line (crack crew) (26 pts)
(b) Isis	50 gun Ship of the Line (crack crew) (17 pts)
(b) Superb	74 gun Ship of the Line (crack crew) (27 pts)
(b) Burford	74 gun Ship of the Line (average crew) (24 pts)

(f) Flamband	50 gun Ship of the Line (average crew) (14 pts)
(f) Annibal	74 gun Ship of the Line (average crew) (24 pts)
(f) Severe	64 gun Ship of the Line (average crew) (18 pts)
(f) Brilliant	80 gun Ship of the Line (crack crew) (31 pts)
(f) Sphinx	80 gun Ship of the Line (average crew) (27 pts)

Nymphe vs. Cleopatre:

Wind from the S, blowing a fresh breeze.

(b) Nymphe	36 gun Frigate (crack crew) (11 pts)
(f) Cleopatre	36 gun Frigate (average crew) (10 pts)

Mars vs. Hercule:

Wind from the S, blowing a fresh breeze.

(b) Mars	74 gun Ship of the Line (crack crew) (26 pts)
(f) Hercule	74 gun Ship of the Line (average crew) (23 pts)

Ambuscade vs. Baionnaise:

Wind from the N, blowing a fresh breeze.

(b) Ambuscade	32 gun Frigate (average crew) (9 pts)
(f) Baionnaise	24 gun Corvette (average crew) (9 pts)

Constellation vs. Insurgent:

Wind from the S, blowing a gale.

(a) Constellation	38 gun Corvette (elite crew) (17 pts)
(f) Insurgent	36 gun Corvette (average crew) (11 pts)

Constellation vs. Vengeance:

Wind from the S, blowing a fresh breeze.

(a) Constellation	38 gun Corvette (elite crew) (17 pts)
(f) Vengeance	40 gun Frigate (average crew) (15 pts)

The Battle of Lissa:

Wind from the S, blowing a fresh breeze.

(b) Amphion	32 gun Frigate (elite crew) (13 pts)
(b) Active	38 gun Frigate (elite crew) (18 pts)
(b) Volage	22 gun Frigate (elite crew) (11 pts)
(b) Cerberus	32 gun Frigate (elite crew) (13 pts)
(f) Favorite	40 gun Frigate (average crew) (15 pts)
(f) Flore	40 gun Frigate (average crew) (15 pts)
(f) Danae	40 gun Frigate (crack crew) (17 pts)
(f) Bellona	32 gun Frigate (green crew) (9 pts)
(f) Corona	40 gun Frigate (green crew) (12 pts)
(f) Carolina	32 gun Frigate (green crew) (7 pts)

Constitution vs. Guerriere:

Wind from the SW, blowing a gale.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Guerriere	38 gun Frigate (crack crew) (15 pts)

United States vs. Macedonian:

Wind from the S, blowing a fresh breeze.

(a) United States	44 gun Frigate (elite crew) (24 pts)
(b) Macedonian	38 gun Frigate (crack crew) (16 pts)

Constitution vs. Java:

Wind from the S, blowing a fresh breeze.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Java	38 gun Corvette (crack crew) (19 pts)

Chesapeake vs. Shannon:

Wind from the S, blowing a fresh breeze.

(a) Chesapeake	38 gun Frigate (average crew) (14 pts)
(b) Shannon	38 gun Frigate (elite crew) (17 pts)

The Battle of Lake Erie:

Wind from the S, blowing a light breeze.

(a) Lawrence	20 gun Sloop (crack crew) (9 pts)
(a) Niagara	20 gun Sloop (elite crew) (12 pts)
(b) Lady Prevost	13 gun Brig (crack crew) (5 pts)
(b) Detroit	19 gun Sloop (crack crew) (7 pts)
(b) Q. Charlotte	17 gun Sloop (crack crew) (6 pts)

Wasp vs. Reindeer:

Wind from the S, blowing a light breeze.

(a) Wasp	20 gun Sloop (elite crew) (12 pts)
(b) Reindeer	18 gun Sloop (elite crew) (9 pts)

Constitution vs. Cyane and Levant:

Wind from the S, blowing a moderate breeze.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Cyane	24 gun Sloop (crack crew) (11 pts)
(b) Levant	20 gun Sloop (crack crew) (10 pts)

Pellew vs. Droits de L'Homme:

Wind from the N, blowing a gale.

(b) Indefatigable	44 gun Frigate (elite crew) (14 pts)
(b) Amazon	36 gun Frigate (crack crew) (14 pts)
(f) Droits L'Hom	74 gun Ship of the Line (average crew) (24 pts)

Algeciras:

Wind from the SW, blowing a moderate breeze.

(b) Caesar	80 gun Ship of the Line (crack crew) (31 pts)
(b) Pompee	74 gun Ship of the Line (crack crew) (27 pts)
(b) Spencer	74 gun Ship of the Line (crack crew) (26 pts)
(b) Hannibal	98 gun 3 Decker SOL (crack crew) (28 pts)
(s) Real-Carlos	112 gun 3 Decker SOL (green crew) (27 pts)
(s) San Fernando	96 gun 3 Decker SOL (green crew) (24 pts)
(s) Argonauta	80 gun Ship of the Line (green crew) (23 pts)
(s) San Augustine	74 gun Ship of the Line (green crew) (20 pts)
(f) Indomptable	80 gun Ship of the Line (average crew) (27 pts)
(f) Desaix	74 gun Ship of the Line (average crew) (24 pts)

Lake Champlain:

Wind from the N, blowing a fresh breeze.

(a) Saratoga	26 gun Sloop (crack crew) (12 pts)
(a) Eagle	20 gun Sloop (crack crew) (11 pts)
(a) Ticonderoga	17 gun Sloop (crack crew) (9 pts)
(a) Preble	7 gun Brig (crack crew) (4 pts)
(b) Confiance	37 gun Frigate (crack crew) (14 pts)
(b) Linnet	16 gun Sloop (elite crew) (10 pts)
(b) Chubb	11 gun Brig (crack crew) (5 pts)

Last Voyage of the USS President:

Wind from the N, blowing a fresh breeze.

(a) President	44 gun Frigate (elite crew) (24 pts)
(b) Endymion	40 gun Frigate (crack crew) (17 pts)
(b) Pomone	44 gun Frigate (crack crew) (20 pts)
(b) Tenedos	38 gun Frigate (crack crew) (15 pts)

Hornblower and the Natividad:

Wind from the E, blowing a gale.

A scenario for you Horny fans. Remember, he sank the Natividad against heavy odds and winds. Hint: don't try to board the Natividad, her crew is much bigger, albeit green.

(b) Lydia	36 gun Frigate (elite crew) (13 pts)
(s) Natividad	50 gun Ship of the Line (green crew) (14 pts)

Curse of the Flying Dutchman:

Wind from the S, blowing a fresh breeze.

Just for fun, take the Piece of cake.

(s) Piece of Cake	24 gun Corvette (average crew) (9 pts)
(f) Flying Dutchy	120 gun 3 Decker SOL (elite crew) (43 pts)

The South Pacific:

Wind from the S, blowing a strong breeze.

(a) USS Scurvy	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(b) HMS Tahiti	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Australian	32 gun Frigate (average crew) (9 pts)
(f) Bikini Atoll	7 gun Brig (crack crew) (4 pts)

Hornblower and the battle of Rosas

Wind from the E, blowing a fresh breeze.

The only battle Hornblower ever lost. He was able to dismast one ship and stern rake another's though. See if you can do as well.

(b) Sutherland	74 gun Ship of the Line (crack crew) (26 pts)
(f) Turenne	80 gun 3 Decker SOL (average crew) (27 pts)
(f) Nightmare	74 gun Ship of the Line (average crew) (24 pts)
(f) Paris	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Napolean	74 gun Ship of the Line (green crew) (20 pts)

Cape Horn:

Wind from the NE, blowing a strong breeze.

(a) Concord	80 gun Ship of the Line (average crew) (27 pts)
(a) Berkeley	98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Thames	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Madrid	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Musket	80 gun 3 Decker SOL (average crew) (27 pts)

New Orleans:

Wind from the SE, blowing a fresh breeze.

Watch that little Cypress go!

(a) Alligator	120 gun 3 Decker SOL (elite crew) (43 pts)
(b) Firefly	74 gun Ship of the Line (crack crew) (27 pts)
(b) Cypress	44 gun Frigate (elite crew) (14 pts)

Botany Bay:

Wind from the N, blowing a fresh breeze.

(b) Shark	64 gun Ship of the Line (average crew) (18 pts)
(f) Coral Snake	44 gun Corvette (elite crew) (24 pts)
(f) Sea Lion	44 gun Frigate (elite crew) (24 pts)

Voyage to the Bottom of the

Wind from the NW, blowing a fresh breeze.

This one is dedicated to David Hedison.

(a) Seaview	120 gun 3 Decker SOL (elite crew) (43 pts)
(a) Flying Sub	40 gun Frigate (crack crew) (17 pts)
(b) Mermaid	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(s) Giant Squid	112 gun 3 Decker SOL (green crew) (27 pts)

Frigate Action:

Wind from the E, blowing a fresh breeze.

(a) Killdeer	40 gun Frigate (average crew) (15 pts)
(b) Sandpiper	40 gun Frigate (average crew) (15 pts)
(s) Curlew	38 gun Frigate (crack crew) (16 pts)

The Battle of Midway:

Wind from the E, blowing a moderate breeze.

(a) Enterprise	80 gun Ship of the Line (crack crew) (31 pts)
(a) Yorktown	80 gun Ship of the Line (average crew) (27 pts)
(a) Hornet	74 gun Ship of the Line (average crew) (24 pts)
(f) Akagi	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Kaga	96 gun 3 Decker SOL (green crew) (24 pts)
(f) Soryu	80 gun Ship of the Line (green crew) (23 pts)

EXAMPLE OF MOVE:

```
      / Max distance (including turns)
     /  Max number of 45 degree turns (one at a time only)
    / /
Move(3, 2): r1l    /* move right, ahead, left
```

```
*
*      0 START
*     b
*--------------------------
*
*     b0 RIGHT
*--------------------------
*
*    b0  ONE
*--------------------------
*     0
*    b   LEFT
*--------------------------
```

SAMPLE GAME:

```
tutorial% sail
Choose a scenario:


NUMBER    SHIPS   IN PLAY   TITLE
0):       2       no        Ranger vs. Drake
1):       2       no        The Battle of Flamborough Head
2):       10      no        Arbuthnot and Des Touches
3):       10      no        Suffren and Hughes
4):       2       no        Nymphe vs. Cleopatre
5):       2       no        Mars vs. Hercule
6):       2       no        Ambuscade vs. Baionnaise
7):       2       no        Constellation vs. Insurgent
8):       2       no        Constellation vs. Vengeance
9):       10      no        The Battle of Lissa
10):      2       no        Constitution vs. Guerriere
11):      2       no        United States vs. Macedonian
12):      2       no        Constitution vs. Java
13):      2       no        Chesapeake vs. Shannon
14):      5       no        The Battle of Lake Erie
15):      2       no        Wasp vs. Reindeer
16):      3       no        Constitution vs. Cyane and Levant
17):      3       no        Pellew vs. Droits de L'Homme
18):      10      no        Algeciras
19):      7       no        Lake Champlain
20):      4       no        Last Voyage of the USS President
21):      2       no        Hornblower and the Natividad
22):      2       no        Curse of the Flying Dutchman
23):      4       no        The South Pacific
24):      5       no        Hornblower and the battle of Rosas bay
25):      5       no        Cape Horn
26):      3       no        New Orleans
27):      3       no        Botany Bay
28):      4       no        Voyage to the Bottom of the Sea
29):      3       no        Frigate Action
30):      6       no        The Battle of Midway

Scenario number? 21
```

```
Your ship is the Lydia, a 36 gun Frigate (elite crew).
Your name, Captain? Dave #1

Initial broadside left (grape, chain, round, double): d

Initial broadside right (grape, chain, round, double): r

Class 3 (36 guns) Frigate 'Lydia' (b0)      Points: 0  Fouls: 0  Grapples: 0
--------------------------------------------------------------------------------
|                                                                              |
|                                                                              |
|                                   ~0 -- a sinking ship                        |
|                  0                                                           |
|                  b                #1 -- an exploding ship                     |
|                  ^                                                           |
|            bow of Lydia                                       |-----------    |
|                                                               | wind speed -5+|
|                                                               | and direction |
|                                                               | (blowing from right)
|            !  -- a struck ship     S0                         |-----------    |
|      1                                           \                           |
|                                                       stern of Natividad     |
|                                                  Natividad has full sails set.
|                                                                              |
--------------------------------Turn 0------------------------------------------
Aye aye, Sir          load: port and starboard --     Load  D! R!  0 1(1)
~                                                     Hull  9           \|/
                        crew: 3 sections ----------   Crew  4  42       -^-     1(1)
                        guns: port and starboard --   Guns  4  4        /|\     e
                      carronades: port and starboard - Carr  2  2        |       3(5)
                        rigging 4 masts -----------   Rigg 5 5 5 5              2(4)
```

NAME

snake, snscore – display chase game

SYNOPSIS

/usr/games/snake [**–w***n*] [**–l***n*]
/usr/games/snscore

DESCRIPTION

Snake is a display-based game which must be played on a CRT terminal from among those supported by vi(1). The object of the game is to make as much money as possible without getting eaten by the snake. The **–l** and **–w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as vi(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc These keys are like hjkl but form a directed pad around the d key.

HJKL These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.

SEFC Likewise for the upper case versions on the left.

ATPB These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, e.g. P is at the far right of the keyboard.

x This lets you quit the game at any time.

p Points in a direction you might want to go.

w Space warp to get out of tight squeezes, at a price.

! Shell escape

^Z Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run */usr/games/snscore* .

FILES

/usr/games/lib/snakerawscores database of personal bests
/usr/games/lib/snake.log log of games played

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

trek – trekkie game

SYNOPSIS

/usr/games/trek [[**–a**] file]

DESCRIPTION

Trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **–a** flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commadore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

COMMAND SUMMARY

abandon	**ca**pture
cloak **u**p/**d**own	
computer request; ...	**da**mages
destruct	**dock**
help	**i**mpulse course distance
lrscan	**m**ove course distance
phasers **a**utomatic amount	
phasers **m**anual amt1 course1 spread1 ...	
torpedo course [**y**es] angle/**n**o	
ram course distance	**r**est time
shell	**sh**ields **u**p/**d**own
srscan [**y**es/**n**o]	
status	**terminate** **y**es/**n**o
undock	**v**isual course
warp warp_factor	

NAME

worm – Play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In *worm,* you are a little worm, your body is the "o"'s on the screen and your head is the "@". You move with the hjkl keys (as in the game snake). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit, if your worm eats the digit is will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms — animate worms on a display terminal

SYNOPSIS

/usr/games/worms [**−field**] [**−length #**] [**−number #**] [**−trail**]

DESCRIPTION

Brian Horn (cithep!bdh) showed me a *TOPS-20* program on the DEC-2136 machine called *WORM*, and suggested that I write a similar program that would run under *Unix*. I did, and no apologies.

−field makes a "field" for the worm(s) to eat; **−trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

SEE ALSO

Snails, by Karl Heuer

BUGS

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump – the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

Wump plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

NAME

miscellaneous — miscellaneous useful information pages

DESCRIPTION

This section contains miscellaneous documentation, mostly in the area of text processing macro packages for *troff*(1).

ascii	map of ASCII character set
eqnchar	special character definitions for eqn
hier	file system hierarchy
mailaddr	mail addressing description
man	macros to typeset manual pages
me	macros for formatting papers
ms	macros for formatting manuscripts

NAME

ascii – map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel|
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb|
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us |
|040 sp |041  ! |042  " |043  # |044  $ |045  % |046  & |047  ´ |
|050  ( |051  ) |052  * |053  + |054  , |055  - |056  . |057  / |
|060  0 |061  1 |062  2 |063  3 |064  4 |065  5 |066  6 |067  7 |
|070  8 |071  9 |072  : |073  ; |074  < |075  = |076  > |077  ? |
|100  @ |101  A |102  B |103  C |104  D |105  E |106  F |107  G |
|110  H |111  I |112  J |113  K |114  L |115  M |116  N |117  O |
|120  P |121  Q |122  R |123  S |124  T |125  U |126  V |127  W |
|130  X |131  Y |132  Z |133  [ |134  \ |135  ] |136  ^ |137  _ |
|140  ` |141  a |142  b |143  c |144  d |145  e |146  f |147  g |
|150  h |151  i |152  j |153  k |154  l |155  m |156  n |157  o |
|160  p |161  q |162  r |163  s |164  t |165  u |166  v |167  w |
|170  x |171  y |172  z |173  { |174  | |175  } |176  ~ |177 del|
```

```
| 00 nul| 01 soh| 02 stx| 03 etx| 04 eot| 05 enq| 06 ack| 07 bel|
| 08 bs | 09 ht | 0a nl | 0b vt | 0c np | 0d cr | 0e so | 0f si |
| 10 dle| 11 dc1| 12 dc2| 13 dc3| 14 dc4| 15 nak| 16 syn| 17 etb|
| 18 can| 19 em | 1a sub| 1b esc| 1c fs | 1d gs | 1e rs | 1f us |
| 20 sp | 21  ! | 22  " | 23  # | 24  $ | 25  % | 26  & | 27  ´ |
| 28  ( | 29  ) | 2a  * | 2b  + | 2c  , | 2d  - | 2e  . | 2f  / |
| 30  0 | 31  1 | 32  2 | 33  3 | 34  4 | 35  5 | 36  6 | 37  7 |
| 38  8 | 39  9 | 3a  : | 3b  ; | 3c  < | 3d  = | 3e  > | 3f  ? |
| 40  @ | 41  A | 42  B | 43  C | 44  D | 45  E | 46  F | 47  G |
| 48  H | 49  I | 4a  J | 4b  K | 4c  L | 4d  M | 4e  N | 4f  O |
| 50  P | 51  Q | 52  R | 53  S | 54  T | 55  U | 56  V | 57  W |
| 58  X | 59  Y | 5a  Z | 5b  [ | 5c  \ | 5d  ] | 5e  ^ | 5f  _ |
| 60  ` | 61  a | 62  b | 63  c | 64  d | 65  e | 66  f | 67  g |
| 68  h | 69  i | 6a  j | 6b  k | 6c  l | 6d  m | 6e  n | 6f  o |
| 70  p | 71  q | 72  r | 73  s | 74  t | 75  u | 76  v | 77  w |
| 78  x | 79  y | 7a  z | 7b  { | 7c  | | 7d  } | 7e  ~ | 7f del|
```

FILES

/usr/pub/ascii

NAME

eqnchar − special character definitions for eqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] **| troff** [options]

neqn /usr/pub/eqnchar [files] **| nroff** [options]

DESCRIPTION

Eqnchar contains *troff* and *nroff* character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with *eqn* and *neqn*. It contains definitions for the following characters

ciplus	⊕	*||*	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	∼	*rangle*	⟩	*blot*	▣
-wig	≃	*hbar*	ℏ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<->*	↔	*empty*	∅
=wig	≅	*<=>*	⇔	*member*	∈
star	*	*|<*	≮	*nomem*	∉
bigstar	✱	*|>*	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	∟	*incl*	⊑
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°		

FILES

/usr/pub/eqnchar

SEE ALSO

troff(1), eqn(1)

NAME

hier — file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative directory hierarchy.

/	root	
/vmunix		
	the kernel binary (UNIX itself)	
/lost+found		
	directory for connecting detached files for *fsck*(8)	
/dev/	devices, see section 4	
	console	main console, *tty*(4)
	tty*	terminals, *tty*(4)
	xy*	disks, *xy*(4S)
	rxy*	raw disks, *xy*(4S)
	...	
/bin/	utility programs, cf /usr/bin/ (described in sect. 1)	
	as	assembler
	cc	C compiler executive, cf /lib/ccom, /lib/cpp, /lib/c2
	csh	C shell
	...	
/lib/	object libraries and other stuff, cf /usr/lib/	
	libc.a	system calls, standard I/O, etc. (described in sect. 2, 3, 3S, 3C, 3N)
	...	
	ccom	C compiler proper
	cpp	C preprocessor
	c2	C code improver
	...	
/etc/	essential data and maintenance utilities; described in section 8	
	dump	dump program *dump*(8)
	passwd	password file, *passwd*(5)
	group	group file, *group*(5)
	motd	message of the day, *login*(1)
	termcap	
		description of terminal capabilities, *termcap*(5)
	ttytype	table of what kind of terminal is on each port, *ttytype*(5)
	mtab	mounted file table, *mtab*(5)
	dumpdates	
		dump history, *dump*(8)
	fstab	file system configuration table *fstab*(5)
	ttys	properties of terminals, *ttys*(5)
	getty	part of *login*, *getty*(8)
	init	the parent of all processes, *init*(8)
	rc	shell script to bring the system up
	cron	the clock daemon, *cron*(8)
	mount	*mount*(8)
	wall	*wall*(1)
	...	
/tmp/	temporary files, cf /usr/tmp/	
	e*	used by *ed*(1)
	ctm*	used by *cc*(1)
	...	
/usr/	general-pupose directory, usually a mounted file system	
	adm/	administrative information

```
        wtmp    login history, utmp(5)
        messages
                hardware error messages
bin/    utility programs, to keep /bin/ small
etc/    administrative programs, to keep /etc/ small
tmp/    temporaries, to keep /tmp/ small
        stm*    used by sort(1)
dict/   word lists, etc.
        words   principal word list, used by look(1)
        spellhist
                history file for spell(1)
games/
        hangman
        lib/    library of stuff for the games
                quiz.k/ what quiz(6) knows
                        index   category index
                        africa  countries and capitals
                        ...
                ...
        ...
include/
        standard #include files
        a.out.h object file layout, a.out(5)
        stdio.h standard I/O, stdio(3S)
        math.h  (3M)
        ...
        sys/    system-defined layouts, cf /sys/h
lib/    object libraries and stuff, to keep /lib/ small
        atrun   scheduler for at(1)
        lint/   utility files for lint
                lint[12] subprocesses for lint(1)
                llib-lc  dummy declarations for /lib/libc.a, used by lint(1)
                llib-lm  dummy declarations for /lib/libc.m
                ...
        struct/ passes of struct(1)
        ...
        tmac/   macros for troff(1)
                tmac.an
                        macros for man(7)
                tmac.s  macros for ms(7)
                ...
        font/   fonts for troff(1)
                ftR     Times Roman
                ftB     Times Bold
                ...
        uucp/   programs and data for uucp(1C)
                L.sys   remote system names and numbers
                uucico  the real copy program
                ...
        units   conversion tables for units(1)
        eign    list of English words to be ignored by ptx(1)
```

```
/usr/
          man/    Pages for major manuals — User's Guide to Commands, UNIX Programmer's
                  Manual, and System Manager's Guide.  man(1)
                  man0/   general
                          intro   introduction to Sun System Manuals, in ms(7) format
                          xx      template for manual page
                  man1/   chapter 1
                          as.1
                          spline.1g
                          ...
                  ...
                  cat1/   preformatted pages for section 1
                  ...
          preserve/
                  editor temporaries preserved here after crashes/hangups
          spool/  delayed execution files
                  at/     used by at(1)
                  lpd/    used by lpr(1)
                          lock    present when line printer is active
                          cf*     copy of file to be printed, if necessary
                          df*     daemon control file, lpd(8)
                          tf*     transient control file, while lpr is working
                  uucp/   work files and staging area for uucp(1C)
                          LOGFILE
                                  summary log
                          LOG.*   log file for one transaction
                          ...
                  mail/   mailboxes for mail(1)
                          name    mail file for user name
                          name.lock
                                  lock file while name is receiving mail
                  secretmail/
                          like mail/
                  wd      initial working directory of a user, typically wd is the user's login name
                          .profile set environment for sh(1), environ(5)
                          .cshrc  startup file for csh(1)
                          .exrc   startup file for ex(1)
                          .mailrc startup file for mail(1)
                          calendar
                                  user's datebook for calendar(1)
                  ucb/    binaries of programs developed at University of California at Berkeley.
                          ...
                          edit    editor for beginners
                          ex      command editor for experienced users
                          ...
                          mail    mail reading/sending subsystem
                          man     on line documentation
                          ...
                          pi      Pascal translator
                          px      Pascal interpreter
                          ...
                          vi      visual editor
```

SEE ALSO

ls(1), whatis(1), whereis(1), which (1), ncheck(8), find(1), grep(1)

BUGS

The position of files is subject to change without notice.

NAME

mailaddr – mail addressing description

DESCRIPTION

Mail addresses are based on the ARPANET protocols listed at the end of this manual page. These addresses are in the general format

user@domain

where a domain is a hierarchical dot separated list of subdomains. For example, the address

eric@monet.Berkeley.ARPA

is normally interpreted from right to left: the message should go to the ARPA name tables (which do not correspond exactly to the physical ARPANET), then to the Berkeley gateway, after which it should go to the local host monet. When the message reaches monet it is delivered to the user "eric".

Unlike some other forms of addressing, this does not imply any routing. Thus, although this address is specified as an ARPA address, it might travel by an alternate route if that was more convenient or efficient. For example, at Berkeley the associated message would probably go directly to monet over the Ethernet rather than going via the Berkeley ARPANET gateway.

Abbreviation. Under certain circumstances it may not be necessary to type the entire domain name. In general anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on "calder.Berkeley.ARPA" could send to "eric@monet" without adding the ".Berkeley.ARPA" since it is the same on both sending and receiving hosts.

Certain other abbreviations may be permitted as special cases. For example, at Berkeley ARPANET hosts can be referenced without adding the ".ARPA" as long as their names do not conflict with a local host name.

Compatibility. Certain old address formats are converted to the new format to provide compatibility with the previous mail system. In particular,

host:user

is converted to

user@host

to be consistent with the *rcp*(1C) command.

Also, the syntax:

host!user

is converted to:

user@host.UUCP

This is normally converted back to the "host!user" form before being sent on for compatibility with older UUCP hosts.

The current implementation is not able to route messages automatically through the UUCP network. This feature is planned for the 4.2 release. Until that time you must explicitly tell the mail system which hosts to send your message through to get to your final destination.

Case Distinctions. Domain names (i.e., anything after the "@" sign) may be given in any mixture of upper and lower case with the exception of UUCP hostnames. Most hosts accept any mixture of case in user names, with the notable exception of MULTICS sites.

Differences with ARPA Protocols. Although the UNIX addressing scheme is based on the ARPA mail addressing protocols, there are some significant differences.

At the time of this writing the only "top level" domain defined by ARPA is the ".ARPA" domain itself. This is further restricted to having only one level of host specifier. That is, the only addresses that ARPA accepts at this time must be in the format "user@host.ARPA" (where "host" is one word). In particular, addresses such as:

eric@monet.Berkeley.ARPA

are not currently legal under the ARPA protocols. For this reason, these addresses are converted to a different format on output to the ARPANET, typically:

eric%monet@Berkeley.ARPA

Route-addrs. Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this routing is done automatically, but sometimes it is desirable to route the message manually. An address that shows these relays are termed "route-addrs." These use the syntax:

<@hosta,@hostb:user@hostc>

This specifies that the message should be sent to hosta, from there to hostb, and finally to hostc. This path is forced even if there is a more efficient path to hostc.

Route-addrs occur frequently on return addresses, since these are generally augmented by the software at each host. It is generally possible to ignore all but the "user@host" part of the address to determine the actual sender.

Postmaster. Every site is required to have a user or user alias designated "postmaster" to which problems with the mail system may be addressed.

CSNET. Messages to CSNET sites can be sent to "user.host@UDel-Relay".

BERKELEY

The following comments apply only to the Berkeley environment.

Host Names. Many of the old familiar host names are being phased out. In particular, single character names as used in Berknet are incompatible with the larger world of which Berkeley is now a member. For this reason the following names are being obsoleted. You should notify any correspondents of your new address as soon as possible.

OLD	NEW	j ingvax	ucbingres
p	ucbcad	r arpavax	ucbarpa
v csvax	ucbernie		
n	ucbkim	y	ucbcory

The old addresses will be rejected as unknown hosts sometime in the near future.

What's My Address? If you are on a local machine, say monet, your address is

yourname@monet.Berkeley.ARPA

However, since most of the world does not have the new software in place yet, you will have to give correspondents slightly different addresses. From the ARPANET, your address would be:

yourname%monet@Berkeley.ARPA

From UUCP, your address would be:

ucbvax!monet.yourname

Computer Center. The Berkeley Computer Center is in a subdomain of Berkeley so that they may administer their own name space. Messages to the computer center should be addressed to one of:

user@host.CC.Berkeley.ARPA
user%host.CC@Berkeley.ARPA

depending on where the message is being sent from. The ".Berkeley.ARPA" may be omitted if the message is sent from inside Berkeley.

For the time being Computer Center hosts are known within the Berkeley domain, i.e., the ".CC" is optional. However, it is likely that this situation will change with time as both the Computer Science department and the Computer Center grow.

Bitnet. Hosts on bitnet may be accessed using:

```
user@host.BITNET
```

This works from 4.2 machines.

NAME

man − macros to typeset manual

SYNOPSIS

nroff −man file ...

troff −man file ...

DESCRIPTION

These macros are used to lay out pages of this manual. The best way to learn how to use the macros is to take an existing manual page and hack it over to your own requirements.

Any text argument *t* may be zero to six words. Quotes may be used to include blanks in a 'word'. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way .I may be used to italicize a whole line, or .SM followed by .B to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **−man**:

*R	'®', '(Reg)' in *nroff*.
*S	Change to default type size.

FILES

/usr/lib/tmac/tmac.an

SEE ALSO

troff(1), nroff(1), man(1)

BUGS

Relative indents don't nest.

REQUESTS

Request	Cause	If no	Explanation
	Break	Argument	
.B *t*	no	*t*=n.t.l.*	Text *t* is bold.
.BI *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and Roman.
.DT	no	.5i 1i...	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Set prevailing indent to *i*. Begin paragraph with hanging indent.
.I *t*	no	*t*=n.t.l.	Text *t* is italic.
.IB *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as .TP with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and Roman.
.LP	yes	-	Same as .PP.
.PD *d*	no	*d*=.4v	Interparagraph distance is *d*.
.PP	yes	-	Begin paragraph. Set prevailing indent to .5i.
.RE	yes	-	End of relative indent. Set prevailing indent to amount of starting .RS.
.RB *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and bold.
.RI *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, move left margin in distance *i*. Set prevailing indent to .5i for nested indents.
.SH *t*	yes	*t*=n.t.l.	Subhead.
.SM *t*	no	*t*=n.t.l.	Text *t* is small.
.TH *n c x v m*	yes	-	Begin page named *n* of chapter *c; x* is extra commentary, e.g. 'local', for

			page foot center; *v* alters page foot left, e.g. '4th Berkeley Distribution'; *m* alters page head center, e.g. 'Brand X Programmer's Manual'. Set prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Set prevailing indent to *i*. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

NAME

me − macros for formatting papers

SYNOPSIS

nroff −me [options] file ...
troff −me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, however these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

Output of the *eqn, neqn, refer,* and *tbl*(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1)
−me Reference Manual, Eric P. Allman
Writing Papers with Nroff Using −me

REQUESTS

In the following list, "initialization" refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The −me Reference Manual* for interesting details.

Request	Initial	Cause	Explanation
	Value	Break	
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x *x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, e.g., abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA**

			(appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *'x'y'z'*	''''	no	Set even footer to x y z
.eh *'x'y'z'*	''''	no	Set even header to x y z
.fo *'x'y'z'*	''''	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *'x'y'z'*	''''	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x;* if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form **.*x**. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *'x'y'z'*	''''	no	Set odd footer to x y z
.oh *'x'y'z'*	''''	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

NAME

ms — text formatting macros

SYNOPSIS

nroff —ms [options] file ...
troff —ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through *col*(1). All external **—ms** macros are defined below.

Note that this **—ms** macro package is an extended version written at Berkeley and is a superset of the standard **— ms** macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many *nroff* and *troff* requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp n	insert n spacing lines
.ce n	center next n lines
.ls n	line spacing: n=1 single, n=2 double space
.na	no alignment of right margin

Font and point size changes with \f and \s are also allowed; for example, "\fIword\fR" will italicize *word*. Output of the *tbl*(1), *eqn*(1) and *refer*(1) preprocessors for equations, tables, and references is acceptable as input.

FILES

/usr/lib/tmac/tmac.s
/usr/lib/ms/ms.???

SEE ALSO

eqn(1), refer(1), tbl(1), troff(1)
Formatting Documents with the —ms Macro Package in
Editing and Text Processing on the Sun Workstation and the
Beginner's Guide to the Sun Workstation.

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	—	y	begin abstract; if *x*=no don't label abstract
.AE	—	y	end abstract
.AI	—	y	author's institution
.AM	—	n	better accent mark definitions
.AU	—	y	author's name
.B *x*	—	n	embolden *x*; if no *x*, switch to boldface
.B1	—	y	begin text to be enclosed in a box
.B2	—	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	—	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	—	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*

.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I,L,C,B; *y*=indent
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for .tl)
.EH *x*	–	n	even page header *x* (3 part as for .tl)
.EN	–	y	end displayed equation produced by *eqn*
.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for .tl)
.OH *x*	–	n	odd page header *x* (3 part as for .tl)
.P1	if TM	n	print header on 1st page
.PP	–	y,y	paragraph with first line indented
.PT	- -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on 1st page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set tabs to 8n 16n ... (nroff) 5n 10n ... (troff)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by *tbl*
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in *troff*
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; *y*=indent
.XE	–	y	end index entry (or series of .IX entries)
.XP	–	y,y	paragraph with first line exdented, others indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page

.2C	–	y,y	begin two column format
.]-	–	n	beginning of *refer* reference
.[0	–	n	end of unclassifiable type of reference
.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in **—ms** by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr LL 6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in **—ms**; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in *nroff,* “ in *troff*)
*U	unquote (" in *nroff,* ” in *troff*)
*–	dash (-- in *nroff,* — in *troff*)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*´	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system bootstrapping, operation and maintenance and describes all the server processes and daemons which run on the system.

Disk formatting and labelling is done by *diag*(8S) which participates in most system bootstraps. Bootstrapping of the system is described in *reboot*(8). The standard set of commands run by the system when it boots is described in *rc*(8). Related commands include ones to check the consistency of file systems *fsck*(8), mount and unmount file systems *mount*(8) and *umount*(8), add swap devices *swapon*(8), cause all outstanding disk I/O to complete *sync*(8), shutdown or reboot a running system *shutdown*(8), *halt*(8), and *reboot*(8), set the time on a machine from the time on another machine *rdate*(8).

Creation of file systems is discussed in *mkfs*(8) and *newfs*(8). File system performance parameters are adjustable with *tunefs*(8). File system saves and restores are described in *dump*(8) and *restore*(8).

Procedures for adding new users to a system are described in *adduser*(8) using *vipw*(8).

Other programs useful when the system crashes or hardware is broken include *gxtest*(8S) which tests the frame buffer on a workstation, *imemtest*(8S) which tests the memory, *crash*(8S) which describes what happens when the system crashes, *savecore*(8) and *analyze*(8) which can be used to analyze system crash dumps. Occasionally useful as adjuncts to the *fsck*(8) file system repair program are *clri*(8), *dcheck*(8), *icheck*(8), and *ncheck*(8).

Configuring a new version of the UNIX kernel requires using the program *config*(8); major system bootstraps often require the use of *mkproto*(8). New devices are made in the /dev directory when device drivers are added to the system by using the *makedev*(8) and *mknod*(8) commands. If you have source, you will use the *install*(8) command to reinstall freshly compiled programs, and *catman*(8) to reformat the pre-formatted version of the manual.

Resource accounting is enabled by the *accton*(8) command, and summarized by *sa*(8). Login time accounting is performed by *ac*(8).

A number of service and daemon processes are described here. The *update*(8) daemon forces delayed disk I/O to occur and *cron*(8) runs periodic events (such as removing temporary files from the disk periodically). The *dmesg*(8) process is invoked by *cron* and keeps the system error log. The *init*(8) process is the initial process created when UNIX boots and manages the reboot process and creates the initial login prompts on the various system terminals through the services of *getty*(8). The Internet super-server *inetd*(8C) invokes all other internet servers as needed. These servers include the remote shell servers *rshd*(8C) and *rexecd*(8C) the remote login server *rlogind*(8C) the FTP and TELNET daemons *ftpd*(8C) and *telnetd*(8C), the TFTP daemon *tftpd*(8C) and the mail arrival notification daemon *comsat*(8C). Other network daemons include the 'load average/who is logged in' daemon *rwhod*(8C), the routing daemon *routed*(8C), and the mail daemon *sendmail*(8).

If network protocols are being debugged, then the protocol debugging trace program *trpt*(8C) is often useful. Remote magnetic tape access is provided by *rsh* and *rmt*(8C). Remote line printer access is provided by *lpd*(8) and control over the various print queues is had through *lpc*(8). Printer cost accounting is done through *pac*(8).

Network host tables may be gotten from the ARPA NIC using *gettable*(8C) and converted to UNIX usable format using *htable*(8).

RCP and NFS daemons include:

/ect/portmap
    used by rpc based services.

/etc/ypbind
used by the yellowpages to locate the yellowpages server.

/etc/biod
used by NFS clients to read ahead to, and write behind from, network file systems.

/ect/nfsd
only on NFS servers, the server's counterpart to */etc/biod*.

/etc/ypserv
implements the yellowpages server, typically on each *nd* server.

/usr/etc/rpc.rstatd
the server counterpart ofthe remote speedometer tools.

/usr/etc.mountd
the server counterpart to *mount*.

/usr/etc/rcp.rwalld
used for broadcasting messages over the network.

LIST OF PROGRAMS

Program	*Appears on Page*	*Description*
ac	ac.8	login accounting
accton	sa.8	system accounting
adbgen	adbgen.8	generate adb script
adduser	adduser.8	procedure for adding new users
analyze	analyze.8	Virtual UNIX postmortem crash analyzer
arp	arp.8c	address resolution display and control
catman	catman.8	create the cat files for the manual
chown	chown.8	change owner
clri	clri.8	clear i-node
comsat	comsat.8c	biff server
config	config.8	build system configuration files
crash	crash.8s	what happens when the system crashes
cron	cron.8	clock daemon
dcheck	dcheck.8	file system directory consistency check
diag	diag.8s	General-purpose stand-alone utility package
dmesg	dmesg.8	collect system diagnostic messages to form error log
dump	dump.8	incremental file system dump
dumpfs	dumpfs.8	dump file system information
expire	expire.8	remove outdated news articles
fastboot	fastboot.8	reboot/halt the system without checking the disks
fasthalt	fastboot.8	reboot/halt the system without checking the disks
fsck	fsck.8	file system consistency check and interactive repair
ftpd	ftpd.8c	DARPA Internet File Transfer Protocol server
gettable	gettable.8c	get NIC format host tables from a host
getty	getty.8	set terminal mode
gxtest	gxtest.8s	stand alone test for the Sun video graphics board
halt	halt.8	stop the processor
htable	htable.8	convert NIC standard format host tables
icheck	icheck.8	file system storage consistency check
ifconfig	ifconfig.8c	configure network interface parameters
imemtest	imemtest.8s	stand alone memory test
inetd	inetd.8c	internet services daemon
init	init.8	process control initialization
iostat	iostat.8	report I/O statistics

kgmon	kgmon.8	generate a dump of the operating system's profile buffers
lpc	lpc.8	line printer control program
lpd	lpd.8	line printer daemon
MAKEDEV	makedev.8	make system special files
makekey	makekey.8	generate encryption key
mkfs	mkfs.8	construct a file system
mknod	mknod.8	build special file
mkproto	mkproto.8	construct a prototype file system
mount	mount.8	mount and dismount file system
ncheck	ncheck.8	generate names from i-numbers
nd	nd.8c	network disk control
netstat	netstat.8	show network status
newaliases	newaliases.8	rebuild the data base for the mail aliases file
newfs	newfs.8	construct a new file system
pac	pac.8	printer/plotter accounting information
pstat	pstat.8	print system facts
quot	quot.8	summarize file system ownership
rc	rc.8	command script for auto-reboot and daemons
rdate	rdate.8	set system date from a remote host
reboot	reboot.8	UNIX bootstrapping procedures
recnews	recnews.8	receive unprocessed articles via mail
renice	renice.8	alter priority of running processes
restore	restore.8	incremental file system restore
rexecd	rexecd.8c	remote execution server
rlogind	rlogind.8c	remote login server
rmail	rmail.8	handle remote mail received via uucp
rmt	rmt.8c	remote magtape protocol module
route	route.8c	manually manipulate the routing tables
routed	routed.8c	network routing daemon
rshd	rshd.8c	remote shell server
rwhod	rwhod.8c	system status server
sa	sa.8	system accounting
savecore	savecore.8	save a core dump of the operating system
sendmail	sendmail.8	send mail over the internet
sendnews	sendnews.8	send news articles via mail
shutdown	shutdown.8	close down the system at a given time
sticky	sticky.8	executable files with persistent text
swapon	swapon.8	specify additional device for paging and swapping
sync	sync.8	update the super block
syslog	syslog.8	log systems messages
telnetd	telnetd.8c	DARPA TELNET protocol server
tftpd	tftpd.8c	DARPA Trivial File Transfer Protocol server
timed	timed.8c	DARPA Time server
trpt	trpt.8c	transliterate protocol trace
tunefs	tunefs.8	tune up an existing file system
umount	mount.8	mount and dismount file system
update	update.8	periodically update the super block
uuclean	uuclean.8c	uucp spool directory clean-up
uurec	uurec.8	receive processed news articles via mail
vipw	vipw.8	edit the password file
vmstat	vmstat.8	report virtual memory statistics

NAME

ac – login accounting

SYNOPSIS

/usr/etc/ac [**–w** wtmp] [**–p**] [**–d**] [people] ...

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login*. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

OPTIONS

–w specifies an alternate *wtmp* file.

–p prints individual totals; without this option, only totals are printed.

–d printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), sa(8), login(1), utmp(5).

NAME

adbgen – generate adb script

SYNOPSIS

/usr/lib/adb/adbgen file.adb ...

DESCRIPTION

Adbgen makes it possible to write *adb*(1) scripts that do not contain hard-coded dependencies on structure member offsets. The input to *adbgen* is a file named *file*.**adb** which contains *adbgen* header information, then a null line, then the name of a structure, and finally an *adb* script. *Adbgen* only deals with one structure per file; all member names are assumed to be in this structure. The output of *adbgen* is an *adb* script in *file*. *Adbgen* operates by generating a C program which determines structure member offsets and sizes, which in turn generates the *adb* script.

The header lines, up to the null line, are copied verbatum into the generated C program. Typically these include C **#include** statements to include the header files containing the relevant structure declarations.

The *adb* script part may contain any valid *adb* commands (see *adb*(1)), and may also contain *adbgen* requests, each enclosed in {}s. Request types are:

1 Print a structure member. The request form is {*member,format*}. *Member* is a member name of the *structure* given earlier, and *format* is any valid *adb* format request. For example, to print the *p_pid* field of the *proc* structure as a decimal number, you would write **{p_pid,d}**.

2 Reference a structure member. The request form is {**member,base*}. *Member* is the member name whose value is desired, and *base* is an *adb* register name which contains the base address of the structure. For example, to get the *p_pid* field of the *proc* structure, you would get the proc structure address in an *adb* register, say <f, and write **{*p_pid,<f}**.

3 Tell *adbgen* that the offset is ok. The request form is **{OFFSETOK}**. This is useful after invoking another *adb* script which moves the *adb* **dot.**

4 Get the size of the *structure*. The request form is **{SIZEOF}**. *Adbgen* replaces this request with the size of the structure. This is useful in incrementing a pointer to step through an array of structures.

5 Get the offset to the end of the structure. The request form is **{END}**. This is useful at the end of the structure to get *adb* to align the **dot** for printing the next structure member.

Adbgen keeps track of the movement of the *adb* **dot** and emits *adb* code to move forward or backward as necessary before printing any structure member in a script. *Adbgen*'s model of the behavior of *adb*'s **dot** is simple: it is assumed that the first line of the script is of the form *struct_address/adb text* and that subsequent lines are of the form +/*adb text*. This causes the *adb* **dot** to move in a sane fashion. *Adbgen* does not check the script to ensure that these limitations are met. *Adbgen* also checks the size of the structure member against the size of the *adb* format code and warns you if they are not equal.

EXAMPLE

If there were an include file x.h which contained:

```
struct x {
        char    *x_cp;
        char    x_c;
        int     x_i;
};
```

Then an *adbgen* file (call it *script.adb*) to print it would be:

```
#include "x.h"

x
./"x_cp"16t"x_c"8t"x_i"n{x_cp,X}{x_c,C}{x_i,D}
```

After running *adbgen* the output file *script* would contain:

```
./"x_cp"16t"x_c"8t"x_i"nXC+D
```

To invoke the script you would type:

```
x$<script
```

DIAGNOSTICS

Warnings about structure member sizes not equal to *adb* format items and complaints about badly formatted requests. The C compiler complains if you reference a structure member that does not exist. It also complains about & before array names; these complaints may be ignored.

FILES

/usr/lib/adb/*	adb scripts for debugging the kernel

SEE ALSO

adb(1), Using ADB to Debug the UNIX Kernel

BUGS

Adb syntax is ugly; there should be a higher level interface for generating scripts.

Structure members which are bit fields cannot be handled because C will not give the address of a bit field. The address is needed to determine the offset.

NAME

adduser — procedure for adding new users

DESCRIPTION

To add a new user account to the system, you must do at least the following things:

1. Create an entry in the system password file for that user.
2. Make a 'home directory' for the user and make sure that the user is the owner of their home directory.
3. Set up some skeletal profile files for the new user.

These steps are described in detail below.

Making an entry in the password file: A new user must choose a login name, which must not already appear in */etc/passwd*. An account can be added by editing a line into the */etc/passwd* file; this must be done with the password file locked, for example, by using *vipw*(8).

A new user is given a group and user id. User id's should be distinct across a system, since they are used to control access to files. Typically, users working on similar projects will be put in the same group. System staff is group '10' for historical reasons, and the super-user is in this group.

A skeletal account for a new user 'francine' would look like:

```
francine::235:20:& Featherstonehaugh:/usr/francine:/bin/csh
```

Fields in the password file have the following meanings:

1. Login name ('francine'). The login name is limited to eight characters in length.
2. Encrypted password. Typically, this field is left empty for a new user so that the first time they log in they don't need a password. They can then set their password to whatever they like using the *passwd*(1) command.
3. User ID. The user ID is a number which identifies that user uniquely in the system. All files that that user creates have this number stored in the data block that describes the file and commands such as *ls*(1) use that number to look in the password file to get the user's name when identifying the owner of the file. For this reason, you cannot just go merrily changing this number at random.
4. Group ID. The group ID is a number which identifies the group to which the user belongs. All files that that user creates have this number stored in the data block that describes the file and commands such as *ls*(1) use that number to look in the groups file to get the groups name when identifying the group ownership of the file.
5. This field is called the 'GCOS' field (from earlier implementation of the UNIX system) and is traditionally used to hold the user's full name. Some installations have other information encoded in this field. From this information we can tell that Francine's real name is 'Francine Featherstonehaugh'. The & here is a shorthand for the user's login name.
6. User's home directory. This is the directory in which that user is 'positioned' when they log in.
7. Initial shell which this user will see on login. If this field is empty, *sh*(1) is used as the initial shell.

Making a home directory for the new user: now you make a home directory for Francine. As shown in the password file entry above, Francine's home directory is */usr/francine*. Francine must be the owner of that directory in order to create files there. So the following sequence would suffice:

```
tutorial# mkdir /usr/francine
tutorial# /etc/chown francine /usr/francine
tutorial#
```

Setting up skeletal profile files: It is useful to give new users some help in getting started, supplying them with a few skeletal files such as *.profile* if they use */bin/sh* as the shell, or *.cshrc* and *.login* if they use */bin/csh* as the shell. New users should be given copies of these files which, for instance, arrange to use *tset*(1) automatically at each login.

FILES

/etc/passwd	password file

SEE ALSO

passwd(1), mkdir(1), chown(8), chsh(1), passwd(5), vipw(8)

NAME

analyze – Virtual UNIX postmortem crash analyzer

SYNOPSIS

/usr/etc/analyze [**–s** swapfile] [**–f**] [**–m**] [**–d**] [**–D**] [**–v**] corefile [system]

DESCRIPTION

Analyze is the post-mortem analyzer for the state of the paging system. In order to use *analyze* you must arrange to get a image of the memory (and possibly the paging area) of the system after it crashes (see *crash*(8S)).

The *analyze* program reads the relevant system data structures from the core image file and indexing information from **/vmunix** (or the specified file) to determine the state of the paging subsystem at the point of crash. It looks at each process in the system, and the resources each is using in an attempt to determine inconsistencies in the paging system state. Normally, the output consists of a sequence of lines showing each active process, its state (whether swapped in or not), its *p0br*, and the number and location of its page table pages. Any pages which are locked while raw i/o is in progress, or which are locked because they are *intransit* are also printed. (Intransit text pages often diagnose as duplicated; you will have to weed these out by hand.)

The program checks that any pages in core which are marked as not modified are, in fact, identical to the swap space copies. It also checks for non-overlap of the swap space, and that the core map entries correspond to the page tables. The state of the free list is also checked.

Options to *analyze*:

–D causes the diskmap for each process to be printed.

–d causes the (sorted) paging area usage to be printed.

–f which causes the free list to be dumped.

–m causes the entire coremap state to be dumped.

–v (long unused) which causes a hugely verbose output format to be used.

In general, the output from this program can be confused by processes which were forking, swapping, or exiting or happened to be in unusual states when the crash occurred. You should examine the flags fields of relevant processes in the output of a *pstat*(8) to weed out such processes.

It is possible to look at the core dump with *adb* if you do

```
adb –k /vmunix /vmcore
```

FILES

/vmunix	default system namelist

SEE ALSO

adb(1S), ps(1), crash(8S), pstat(8)

AUTHORS

Ozalp Babaoglu and William Joy

DIAGNOSTICS

Various diagnostics about overlaps in swap mappings, missing swap mappings, page table entries inconsistent with the core map, incore pages which are marked clean but differ from disk-image copies, pages which are locked or intransit, and inconsistencies in the free list.

It would be nice if this program analyzed the system in general, rather than just the paging system in particular.

NAME

arp – address resolution display and control

SYNOPSIS

arp *hostname*
arp -a [*vmunix* [*kmem*
arp -d *hostname*
arp -s *hostname ether_addr* [**temp**] [**pub**]
arp -f *filename*

DESCRIPTION

The *arp* program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol (*arp*(4p)).

With no flags, the program displays the current ARP entry for *hostname*. With the **-a** flag, the program displays all of the current ARP entries by reading the table from the file *kmem* (default /dev/kmem) based on the kernel file *vmunix* (default /vmunix).

With the **-d** flag, a super-user may delete an entry for the host called *hostname*.

The **-s** flag is given to create an ARP entry for the host called *hostname* with the Ethernet address *ether_addr*. The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless the word **temp** is given in the command. If the word **pub** is given, the entry will be "published", e.g., this system will respond to ARP requests for *hostname* even though the hostname is not its own.

The **-f** flag causes the file *filename* to be read and multiple entries to be set in the ARP tables. Entries in the file should be of the form

> *hostname ether_addr* [**temp**] [**pub**]

with argument meanings as given above.

SEE ALSO

arp(4p), ifconfig(8c)

NAME

catman – create the cat files for the manual

SYNOPSIS

/usr/etc/catman [**–p**] [**–n**] [**–w**] [sections]

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the nroff input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* recreates the **/usr/man/whatis** database.

If there is one parameter not starting with a '–', it is take to be a list of manual sections to look in. For example

```
catman 123
```

only updates manual sections 1, 2, and 3.

OPTIONS

–n Do not create **/usr/man/whatis**.

–p Print what would be done instead of doing it.

–w Only create the **/usr/man/whatis** database. No manual reformatting is done.

FILES

/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/makewhatis	commands to make whatis database

SEE ALSO

man(1)

NAME

chown – change owner

SYNOPSIS

/etc/chown **–f** owner file ...

DESCRIPTION

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

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

OPTIONS

–f Do not report errors.

FILES

/etc/passwd

SEE ALSO

chgrp(1), chown(2), passwd(5), group(5)

NAME

clri − clear i-node

SYNOPSIS

/etc/clri filesystem i-number ...

DESCRIPTION

N.B.: *Clri* is obsoleted for normal file system repair work by *fsck*(8).

Clri writes zeros on the i-nodes with the decimal *i-numbers* on the *filesystem.* After *clri,* any blocks in the affected file will show up as 'missing' in an *icheck*(8) of the *filesystem.*

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

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

SEE ALSO

icheck(8)

BUGS

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

NAME

comsat — biff server

SYNOPSIS

/usr/etc/in.comsat

DESCRIPTION

Comsat is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by *inetd*(8C), and times out if inactive for a few minutes.

Comsat listens on a datagram port associated with the "biff" service specification (see *services*(5)) for one line messages of the form

user@mailbox-offset

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a "biff y"), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the "From", "To", "Date", or "Subject" lines are not printed when displaying the message.

FILES

/etc/utmp to find out who's logged on and on what terminals

SEE ALSO

biff(1)

BUGS

The message header filtering is prone to error.

The notification should appear in a separate window so it does not mess up the screen.

NAME

config — build system configuration files

SYNOPSIS

/etc/config [**−p**] *config_file*

DESCRIPTION

Config does the preparation necessary for building a new system kernel with *make*(1). The *config_file* named on the command line describes the kernel to be made in terms of options you want in your system, size of tables, and device drivers to be included. The format of this file is described below. Using the **−p** option on the command line configures a system for profiling (see *kgmon*(8) and *gprof*(1)).

When you run *config*, it uses several input files located in the *conf* subdirectory of the system source (including your *config_file*) to generate the files necessary to compile and link your kernel. Since *config* places these output files in the directory named **..**/*config_file* — which it assumes exists — you must create this directory before running *config*. One of *config*'s output files is a *makefile* which you use with *make*(1) to build your system. For a description of other input and output files, see the FILES section below.

You use *config* as follows. Run *config* from the *conf* subdirectory of the system source (in a typical Sun environment, from */sys/conf*):

```
tutorial# /etc/config config_file
Don't forget to run make depend
tutorial#
```

While *config* is running watch for any errors. Never use a kernel which *config* has complained about; the results are unpredictable. If *config* completes successfully, you can change directory to the **..**/*config_file* directory, where it has placed the new *makefile*, and use *make* to create the dependency graph for the new system:

```
Don't forget to run make depend
tutorial# cd ../config_file
tutorial# make depend
 ...lots of output...
tutorial# make
 ...lots more output...
```

Then you can install your new kernel and try it out.

CONFIG FILE FORMAT

In the following descriptions, a number can be a decimal integer, a whole octal number or a whole hexadecimal number. Hex and octal are specified to *config* in the same way they are specified to the C compiler, a number starting with "0x" is a hex number and a number starting with just a "0" is an octal number. When specifying the timezone, you may also use floating point numbers.

Comments are specified in a config file with the character "#". All characters from a "#" to the end of a line are ignored.

Lines beginning with tabs are considered continuations of the previous line.

Lines of the configuration file can be one of two basic types. First, there are lines which describe general things about your system:

machine *type*

This is system is to run on the machine type specified. Only one machine type can appear in the config file. The legal *type* for a Sun system is **sun**.

cpu "*type*"
: This system is to run on the cpu type specified. For a Sun system, "*type*" is **"SUN2"** (note that the double quotes are part of the designation).

ident *name*
: Gives the system identifier — a name for the machine or machines that run this kernel. *name* must be enclosed in double quotes if it contains both letters and numbers (for example, "SDST120"), or you will get a syntax error when you run */etc/config*. Also, note that if *name* is **GENERIC**, you can specify the unique *config_clause* **swap generic** in the **config** line described below (see the next section, *Specific System Description Lines*). If you use any other string for *name*, and you also include an **options GENERIC** line, you can still use the **swap generic** line. However, if you use any other string for *name* and omit the **options GENERIC** line, you may **NOT** use the line **config vmunix swap generic** to specify your kernel image.

timezone *number* [**dst**]
: Specifies the timezone you are in, measured in the number of hours west of GMT. 5 is EST, 8 is PST. Negative numbers indicate hours east of GMT. If you specify **dst,** the system will convert to and from daylight savings time when appropriate.

maxusers *number*
: The maximum expected number of simultaneously active user on this system is *number*. This number is used to size several system data structures.

options *optlist*
: Compile the listed options into the system. Options in this list are separated by commas. There is a list of options that you may specify in the generic makefile. A line of the form "options FUNNY,HAHA" yields −DFUNNY −DHAHA to the C compiler. An option may be given a value, by following its name with "=" then the value enclosed in (double) quotes. None of the standard options use such a value.

config *sysname config_clauses...*
: Generate a system with name *sysname and* configuration as specified in *config-clauses*. The *sysname* is used to name the resultant binary image and per-system swap configuration files. The *config_clauses* indicate the location for the root file system, one or more disk partitions for swapping and paging, and a disk partition to which system dumps should be made. All but the root device specification may be omitted; *config* will assign default values as described below.

root
: A root device specification is of the form **root** *on xy0d*. If a specific partition is omitted — for example, if only **root on xy0** is specified — the "a" partition is assumed. When a generic system is being built, no root specification should be given; the root device will be defined at boot time by prompting the console.

swap
: Swap device specifications have two possible forms. If a generic swap configuration is required, the clause **swap generic** should be specifed. Otherwise, if a single partition is to be used for swapping, one may specify **swap** *on xy0b*. If multiple partitions are to be interleaved one should specify something of the form **swap** *on xy0* **and** *xy1* **and** *xy1g*. If no swap specification is given, *config* assumes swapping should be done on the "b" partition of the root device. Swapping areas may be almost any size and multiple swap partitions of varying size may be interleaved. Partitions used for swapping are sized at boot time by the system; to override dynamic sizing of a swap area the number of sectors in the swap area can be specified in the config file. For example, **swap** *on xy0b size 99999* would configure a swap partition with 99999 sectors.

dumps
: The location to which system dumps are sent may be specified with a clause of the form **dumps** *on xy1*. If no dump device is specified, the first swap partition

specified is used. If a device is specified without a particular partition, the "b" partition is assumed. If a generic configuration is to be built, no dump device should be specified; the dump device will be assigned to the swap device dynamically configured at boot time.

Dumps are placed at the end of the partition specified. Their size and location is recorded in global kernel variables *dumpsize* and *dumplo*, respectively, for use by *savecore*(8).

Device names specified in configuration clauses are mapped to block device major numbers with the file *devices.machine*, where *machine* is the machine type previously specified in the configuration file. If a device name to block device major number mapping must be overridden, a device specification may be given in the form **major** *x* **minor** *y*.

The second group of lines in the configuration file describe which devices your system has and what they are connected to (for example, I have a Xylogics 450 Disk Controller on the main bus). These lines have the following format:

dev_type *dev_name* **at** *con_dev* *more_info*

Dev_type is either **tape, disk, controller, device,** or **pseudo-device.** These types have the following meanings:

controller
is a disk or tape controller.

disk or **tape**
are devices connected to a controller.

device
is something 'attached' to the main system bus, like a cartridge tape interface.

pseudo-device
A software subsystem or driver treated like a device driver, but without any associated hardware. Current examples are the pseudo-tty driver and various network subsystems. (For pseudo-devices, **more_info** may be specified as an integer, that gives the value of the symbol defined in the header file created for that device, and is generally used to indicate the number of instances of the pseudo-device to create. If you load a subsystem you will probably find it convenient to enable conditional code using an **options** specification.

Dev_name is the standard UNIX device name and unit number (if the device is not a **psuedo-device**) of the device you are specifying. For example, **xyc0** is the *dev_name* for the first Xylogics controller in a system; **ar0** names the first quarter-inch tape controller.

Con_dev is what the device you are specifying is connected to. For example, disk **xy1** is connected to controller **xyc0**.

More_info is a sequence of the following:

csr [*bus_spec space_spec*] *addr*
Specifies the address of the csr (command and status registers) for a device. The optional bus specification (*bus_spec*) and space specification (*space_spec*) pair may be included in the address; otherwise, *config* will use a heuristic using *addr* to determine the bus and space. Typically, *addr* is given as a hexidecimal value.

bus_spec may be:

all	The device is on any main system bus.
mb	The device is on the Multibus.
vme	The device is on the VMEbus.

space_spec may be:

virt Virtual address (preset by monitor) follows.
obmem The following address is in on-board memory.
obio The following address is in on-board I/O.
busmem The following address is in main bus memory.
busio The following address is in main bus I/O.

The csr address must be specified for all controllers, and for all devices connected to the main system bus (whether it is a Multibus or a VMEbus).

drive *number*
For a disk or tape, specifies which drive this is.

flags *number*
These flags are made available to the device driver, and are usually read at system initialization time.

priority *level*
For devices which interrupt, specifies the interrupt level at which the device operates.

vector *intr number* [*intr number* . . .]
For devices which use vectored interrupts on VMEbus systems, *intr* specifies the vectored interrupt routine and *number* the corresponding vector to be used (64-255).

A **?** may be substituted for a number in two places and the system will figure out what to fill in for the **?** when it boots. You can put question marks on a *con_dev* (for example, at xyc?), or on a drive number (for example, drive ?). This allows redundancy, as a single system can be built which will boot on different hardware configurations.

The easiest way to understand config files it to look at a working one and modify it to suit your system. Good examples are provided in the *Configuring the System Kernel* chapter and second appendix of the *Installing UNIX on the Sun Workstation* manual.

FILES

Files in */sys/conf* which may be useful for developing the *config_file* used by *config* are:

GENERIC Generic configuration file for Sun systems, containing all possible device description lines
README File describing how to make a new kernel

As shipped from Sun, the files used by */etc/config* as input are in the */sys/conf* directory:

config_file System-specific configuration file
makefile.sun Generic prototype makefile for Sun systems
files List of common files required to build a basic kernel
files.sun List of files for a Sun-specific kernel
devices.sun Name to major device mapping file for Sun systems

/etc/config places its output files in the *../config_file* directory:

mbglue.s Short assembly language routines used for vectored interrupts
ioconf.c Describes I/O devices attached to the system
makefile Used with *make*(1) to build the system
device_name.h A set of header files (various *device_name*'s) containing devices which can be compiled into the system.

SEE ALSO

The SYNOPSIS portion of each device entry in the Section 4 pages of the *System Interface Manual.*

In the *Installing UNIX on the Sun Workstation* Manual, see the *Configuring the System Kernel* chapter, and the *Sample Configuration Files* appendix.

BUGS

The line numbers reported in error messages are usually off by one.

NAME

crash – what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes and how you can analyze crash dumps.

When the system crashes voluntarily it displays a message of the form

> panic: why i gave up the ghost

on the console, takes a dump on a mass storage peripheral, and then invokes an automatic reboot procedure as described in *reboot*(8). Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure the system will then resume multiuser operations.

The system has a large number of internal consistency checks; if one of these fails, it will panic with a very short message indicating which one failed.

The most common cause of system failures is hardware failure, which can reflect itself in different ways. Here are the messages which you are likely to encounter, with some hints as to causes. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

IO err in push
hard IO err in swap

The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

timeout table overflow

This really shouldn't be a panic, but until we fix up the data structure involved, running out of entries causes a crash. If this happens, you should make the timeout table bigger by changing the value of *ncallout* in the *param.c* file, and then rebuild your system.

trap type *type*, **pid** *process-id*, **pc =** *program-counter*, **sr =** *status-register*, **context** *context-number*

A unexpected trap has occurred within the system; typical trap types are:

- Bus error
- Address error
- Illegal instruction
- Divide by zero
- Chk instruction
- Trapv instruction
- Privilege violation
- Trace
- 1010 emulator trap
- 1111 emulator trap
- Stack format error
- Unitialized interrupt
- Spurious interrupt

The favorite trap types in system crashes are 'Bus error' or 'Address error', indicating a wild reference. The *process-id* is the id of the process running at the time of the fault, *program-counter* is the hexadecimal value of the program counter, *status-register* is the hexadecimal value of the status register, and *context-number* is the context that the process was running in. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes.

init died

> The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

That completes the list of panic types you are likely to see.

When the system crashes it writes (or at least attempts to write) an image of memory into the back end of the primary swap area. After the system is rebooted, the program *savecore*(8) runs and preserves a copy of this core image and the current system in a specified directory for later perusal. See *savecore*(8) for details.

To analyze a dump you should begin by running *adb*(1S) with the **−k** flag on the core dump. A more complete discussion of system debugging is impossible here. See, however, 'Using ADB to Debug the UNIX Kernel'.

SEE ALSO

adb(1S), analyze(8), reboot(8)
Using ADB to Debug the UNIX Kernel, in the *System Internals for the Sun Workstation*

NAME

cron — clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

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

/usr/lib/crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=Monday). Each of these patterns may contain a number in the range above; two numbers separated by¯a dash meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the shell at the specified times. A percent character in this field 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.

Here are a few example lines from */usr/lib/crontab*, to give you a better sense of the file's format:

```
0 0 * * * calendar -
15 0 * * * /usr/etc/sa -s >/dev/null
0,30 * * * * /etc/dmesg - >>/usr/adm/messages
15 4 * * * find /usr/preserve -mtime +7 -a -exec rm -f {} ;
10 4 * * * egrep 'SYSERR¦refused¦unreachable' /usr/spool/log/syslog.0 ¦mail Postmaster
```

Cron examines */usr/lib/crontab* under the following conditions:

- At least once per hour (on the hour).
- When the next command is to be run — *cron* looks ahead until the next command and sleeps until then.
- When *cron*'s process is sent a SIGHUP. This means that comeone who changes */usr/lib/crontab* can get *cron* to look at it right away.

You can also create the */usr/adm/cronlog* file, if you wish. If the file exists, *cron* logs to it each time it executes an instruction from */usr/lib/crontab*. You can thus use the *cronlog* file to make sure *cron* is running properly. The lines in the file consist of a timestamp, and process statement. Lines look something like this:

```
Thu Mar  8 10:20:00 1984: /etc/dmesg - >>/usr/adm/messages
Thu Mar  8 10:29:59 1984: /etc/atrun
Thu Mar  8 10:30:00 1984: /etc/dmesg - >>/usr/adm/messages
Thu Mar  8 10:39:59 1984: /etc/dmesg - >>/usr/adm/messages
```

FILES

/usr/lib/crontab	Instruction file
/usr/adm/cronlog	Log file

SEE ALSO

crontab(5)

NAME

dcheck – file system directory consistency check

SYNOPSIS

/usr/etc/dcheck [**–i** *numbers*] [*filesystem*]

DESCRIPTION

N.B.: *Dcheck* is obsoleted for normal consistency checking by *fsck*(8).

Dcheck reads the directories in a file system and compares the link-count in each inode with the number of directory entries by which it is referenced. If the file system is not specified, *dcheck* checks a set of default file systems.

Dcheck is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

OPTIONS

–i *numbers*

Numbers is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), icheck(8), fs(5), clri(8), ncheck(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since *dcheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Dcheck is obsoleted by *fsck* and remains for historical reasons.

Inode numbers less than 2 are invalid.

NAME

diag – stand-alone disk initialization and diagnosis

SYNOPSIS

>b stand/diag

DESCRIPTION

Diag is a general purpose stand-alone package for initializing, diagnosing, and repairing disks. It supports the various SMD and ST-506 disk controllers.

Note: *Diag* is not a system utility and can only be called from the Sun Workstation PROM monitor.

The most common use of *diag* is formatting and labelling a disk — see the *format*, *label*, and *partition* commands.

Immediately after startup, *diag* prompts for information about the particular disk drive(s) and controller(s) it is to work with. There is a facility for specifying additional information about non-standard disks.

Diag tries to access the disk controller you have defined. If the attempt succeeds, it gives you a status report on the disk and issues the 'diag>' prompt. If the attempt fails (if the controller is ill-defined or nonexistent), you get a bus error message, and return to the PROM monitor.

Once configured, *diag* accepts the comands listed below. Only enough characters to uniquely identify the command need be typed.

abortdma Aborts/does not abort the *dmatest* command when a data miscompare is detected. That is, if the internal variable set by *abortdma* is 'on' (default), the *dmatest* command aborts as soon as it finds an error; otherwise, the *dmatest* continues.

clear Sends a restore command to a disk. This is needed to manually reset disk faults.

diag Re-initializes the *diag* program itself — goes back to phase one of the initialization process described above.

dmatest Begins a continuous DMA test. The test copies random data to and from the designated controller, comparing data. If a miscompare is found, an error message is displayed and the test aborts (unless the *abortdma* command is used — see *abortdma*, above). **Note**: this command is available only with the Xylogics 450 Controller.

errors Reports/does not report all errors as they occur (toggles; default is reporting off).

fix Formats and verifies a range of tracks; any defective tracks/sectors found are automatically corrected using mapping or slipping. **Note:** this command is available for SMD controllers only.

format For SMD disks, formats the entire disk; for SCSI disks, initiates the SCSI *format* program.

help or **?** Displays a list of the available commands.

info Reports/suppresses report of all disk activity as it completes (toggles; default is reporting off).

label Labels the disk.

map Displays current mappings and allows you to explicitly map one track/sector to a different track/sector. Usually used for manual bad sector mapping. The *format* and *fix* commands usually do this automatically when a bad track/sector is found. **Note:** the *map* command is disk controller-dependent: you can *map* tracks with an Interphase controller, and sectors with Xylogics controllers. This command is not available for use with Adaptec disk controllers (ST-506 interface).

mapcheck Enables/disables checking for overlapping mapped sectors/tracks during the *position*, *read*, *test*, or *write* commands. If the internal variable set by *mapcheck* is 'on' when an error occurs (default), then the current mappings (if any) are read from the disk and checked to see if there are overlapping mappings over the area of the disk where the transfer failed.

partition Creates, assigns, or modifies logical partition tables for a disk. The UNIX operating system requires logical partitions. The *label* command writes the partition map to the disk. There are standard partition tables for each type of disk that *diag* knows about.

position Continuously tests the disk by reading random sectors from the disk. To abort the test, type a **^C** (CONTROL-C).

quit Quits from *diag* and returns to the PROM monitor.

rhdr Reads and displays the track headers for a specified track. **Note:** this command is available for the Xylogics 450 controller only.

read Reads specified blocks from the disk. The *read* command prompts for the starting block number, number of blocks, and the block increment. The *read* command doesn't report the data it reads — it is intended for verifying that blocks are readable.

scan Continuously scans over a range of sectors looking for defective sectors by writing/reading/verifying various bit patterns to sequential sectors. Any data on the disk in the range to be scanned is destroyed. Sectors previously mapped are not scanned, so any errors reported will be newly found defective sectors. If used with a Xylogics controller, defective sectors can be automatically mapped/slipped when they are found. To abort the *scan*, type a **^C** (CONTROL-C).

seek Performs a seek test on the disk: a seek is made to every cylinder and to every possible cylinder distance.

slip Explicitly slips one sector on a track. Usually used for manual bad sector slipping. The *format* and *fix* commands usually do this automatically when bad sectors are found and the proper conditions exist. **Note:** slipping is available only with the Xylogics 450 controller, and only when the disk has a spare data sector per track.

slipmsgs Displays/suppresses display of track headers before and after each slip operation that occurs during the *fix*, *format*, and *slip* commands (toggles; default is display off).

status Reports the ready status of each drive on the current controller.

test Continuously tests the disk by writing random data to random sectors on the disk and then verifying that the correct data can be read back. The *test* command destroys data on the disk. To abort the *test*, type a **^C** (CONTROL-C).

time Turns timing on/off. When timing is on, *diag* reports on how long certain operations take — *diag* is less verbose in this state so it doesn't waste time displaying messages (toggles; default is timing off).

translate Translates a given block number into its decimal value, hexadecimal value, and logical disk address.

verify Reads and displays the label from the disk. Shows the logical partition assignments. This done automatically when the *label* command has labelled the disk.

version Displays the SCCS identification strings for this version of *diag*.

whdr Modifies and writes the track headers for a specified track. **Note:** this command is available with the Xylogics 450 controller only. Also, it should be used only for specialized patching — misuse may destroy track data.

write Verifies that blocks are writable by writing garbage data to specified blocks on the disk. The *write* command prompts for the starting block number, number of blocks, and the block increment.

+ Adds two block numbers and reports the result in decimal, hexadecimal, and as a logical disk address.

– Subtracts two block numbers and reports the result in decimal, hexadecimal, and as a logical disk address.

Block numbers may be entered either as an absolute decimal block number, or as a disk address of the form cylinder/head/sector.

Any *diag* command may be aborted by typing a **^C** (CONTROL-C).

NAME

dkinfo – report information about a disk's geometry and partitioning

SYNOPSIS

/etc/dkinfo *disk*[*partition*]

DESCRIPTION

Dkinfo gives the total number of cylinders, heads, and sectors or tracks on the specified *disk*, and gives this information along with the starting cylinder for the specified *partition*. If no *partition* is specified on the command line, *dkinfo* reports on all partitions.

The *disk* specification here is a disk name of the form *xxn*, where *xx* is the controller device abbreviation (ip, xy, etc.) and *n* is the disk number. The *partition* specification is simply the letter used to identify that partition in the standard UNIX nomenclature. For example, '/etc/dkinfo xy0' reports on the first disk in a system controlled by a Xylogics controller; '/etc/dkinfo xy0g' reports on the seventh partition of such a disk.

You must be be able to open */dev/rxxnp* to run *dkinfo*. Usually, only the superuser can do so.

EXAMPLE

A request for information on my local disk, an 84 MByte disk controlled by a Xylogics 450 controller, might look like this:

```
# /etc/dkinfo xy0
xy0: Xylogics 450 controller at addr ee40, unit # 0
586  cylinders 7 heads 32 sectors/track
a: 15884 sectors (70 cyls, 6 tracks, 12 sectors)
   starting cylinder 0
b: 33440 sectors (149 cyls, 2 tracks)
   starting cylinder 71
c: 131264 sectors (586 cyls)
   starting cylinder 0
d: No such device or address
e: No such device or address
f: No such device or address
g: 81760 sectors (365 cyls)
   starting cylinder 221
h: No such device or address
#
```

SEE ALSO

dk(4), diag(8)

NAME

dmesg — collect system diagnostic messages to form error log

SYNOPSIS

/usr/etc/dmesg [−]

DESCRIPTION

Dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. If the − flag is given, then *dmesg* computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with *cron*(8) to produce the error log */usr/adm/messages* by running the command

```
/etc/dmesg − >> /usr/adm/messages
```

every 10 minutes.

FILES

/usr/adm/messages	error log (conventional location)
/usr/adm/msgbuf	scratch file for memory of − option

BUGS

The system error message buffer is of small finite size. As *dmesg* is run only every few minutes, not all error messages are guaranteed to be logged. This can be construed as a blessing rather than a curse.

Error diagnostics generated immediately before a system crash will never get logged.

NAME

dump – incremental file system dump

SYNOPSIS

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

DESCRIPTION

Dump copies to magnetic tape all files changed after a certain date in the *filesystem*. The *key* specifies the date and other options about the dump. *Key* consists of characters from the set **0-9¦b¦c¦**

0–9 This number is the 'dump level'. All files modified since the last date stored in the file */etc/dumpdates* for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the entire filesystem is dumped if the **0** option is used.

b Specifies the blocking factor for the dump. The blocking factor is taken from the next argument on the command line. The default blocking factor is 10.

c Dump to cartridge tape instead of standard half-inch magnetic tape.

d The density of the tape, expressed in BPI, is taken from the next *argument*. This is used in calculating the amount of tape used per reel. The default is 1600.

f Place the dump on the next *argument* file instead of the tape. If file is of the form *machine:device*, dump to a remote machine. If *argument* is "–", dump to standard output. If dump is called as **rdump**, the tape defaults to **dumphost:**/*dev/rmt8*, **otherwise, the tape defaults to** /*dev/rmt8*. **Use an alias for dumphost** *in the file /etc/hosts to direct the output to a desired remote machine.*

n Whenever *dump* requires operator attention, notify by means similar to a *wall*(1) all of the operators in the group "operator".

s The size of the dump tape is specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, *dump* waits for reels to be changed. The default tape size is 2300 feet.

u If the dump completes successfully, write the date of the beginning of the dump on file */etc/dumpdates*. This file records a separate date for each filesystem and each dump level. The format of */etc/dumpdates* is readable by people, consisting of one free format record per line: filesystem name, increment level and *ctime(3)* format dump date. */etc/dumpdates* may be edited to change any of the fields, if necessary.

w Dump tells the operator of filesystems which need to be dumped. The information is gleaned from the files */etc/dumpdates* and */etc/fstab*. If the **w** option is set, all other options are ignored, and dump exits immediately.

W Like the **w**, but the **W** option causes *dump* to print out, for each file system in */etc/dumpdates* the most recent dump date and level, and highlights those file systems that should be dumped.

If no arguments are given, the *key* is assumed to be **9u**.

Dump requires operator intervention on these conditions: end of tape, end of dump, tape write error, tape open error or disk read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** key, *dump* interacts with the operator on *dump's* control terminal at times when *dump* can no longer proceed, or if something is grossly wrong. All questions *dump* poses **must** be answered by typing "yes" or "no", appropriately.

Since making a dump involves a lot of time and effort for full dumps, *dump* checkpoints itself at the start of each tape volume. If writing that volume fails for some reason, *dump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

Dump tells the operator what is going on at periodic intervals, including usually low estimates of the number of blocks to write, the number of tapes it will take, the time to completion, and the time to the tape change. The output is verbose, so that others know that the terminal controlling *dump* is busy, and will be for some time.

Now a short suggestion on how to perform dumps. Start with a full level 0 dump

```
dump 0un
```

Next, dumps of active file systems are taken on a daily basis, using a modified Tower of Hanoi algorithm, with this sequence of dump levels:

3 2 5 4 7 6 9 8 9 9 ...

For the daily dumps, a set of 10 tapes per dumped file system is used on a cyclical basis. Each week, a level 1 dump is taken, and the daily Hanoi sequence repeats with 3. For weekly dumps, a set of 5 tapes per dumped file system is used, also on a cyclical basis. Each month, a level 0 dump is taken on a set of fresh tapes that is saved forever.

FILES

/dev/rmt8	default tape unit to dump to
/etc/dumpdates	new format dump date record
/etc/fstab	dump table: file systems and frequency
/etc/group	to find group *operator*

SEE ALSO

restore(8), dump(5), fstab(5)

DIAGNOSTICS

Many, and verbose.

Dump exit codes worthy of note are:

0	normal exit when **w** or **W** options are used.
1	normal exit.
2	error — restart writing from last checkpoint.
3	abort — no checkpoint attempted.

BUGS

Sizes are based on 1600 BPI blocked tape; the raw magtape device has to be used to approach these densities. Fewer than 32 read errors on the filesystem are ignored. Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

It would be nice if *dump* knew about the dump sequence, kept track of the tapes scribbled on, told the operator which tape to mount when, and provided more assistance for the operator running *restore*.

NAME

dumpfs – dump file system information

SYNOPSIS

/usr/etc/dumpfs *device*

DESCRIPTION

Dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO

fs(5), tunefs(8), newfs(8), fsck(8)

NAME

expire — remove outdated news articles

SYNOPSIS

/usr/lib/news/expire [**—n** *newsgroups*] [**—i**] [**—I**] [**—v**[*level*]] [**—e** *days*] [**—a**]
[**—r**] [**—h**]

DESCRIPTION

Expire is normally started up by *cron*(8) every night to remove all expired news. If no newsgroups are specified, the default is to expire **all**.

Articles whose specified expiration date has already passed are considered expirable.

OPTIONS

—n *newsgroups*
List of newsgroups to expire. Elements in the list of newsgroups must be separated by commas and/or spaces.

—a archive articles in /usr/spool/oldnews. Articles are unlinked in the absence of the **—a** option.

—v[*level*]
be more verbose. A verbosity *level* (default 1) can follow the **—v** flag, as in **—v3** for even more output. This is useful if articles aren't being expired and you want to know why.

—e *days*
give the number of days to use for a default expiration date. If not given, an installation dependent default (often 2 weeks) is used. Note that you must use a space between **—e** and *days.*

—i
—I ignore any expiration date explicitly given on articles. This can be used when disk space is really tight. **—I** always ignores expiration dates, while **—i** only ignores the date if ignoring it would expire the article sooner. *WARNING:* If you have articles archived by giving them expiration dates far into the future, these options might remove these files anyway.

—r rebuild the history file without removing any files. In the process, *expire* formats the *dbm*(3) format files associated with the history file.

—h expire articles without using the history file. Both the **—r** and **—h** flags use the active file for newsgroup information rather than the history file.

SEE ALSO

checknews(1), inews(1), readnews(1), recnews(8), sendnews(8), uurec(8)

NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

SYNOPSIS

/etc/fastboot [*boot-options*]
/etc/fasthalt [*halt-options*]

DESCRIPTION

Fastboot and *fasthalt* are shell scripts which reboot and halt the system without checking the file systems. This is done by creating a file */fastboot*, then invoking the *reboot* program. The system startup script, */etc/rc*, looks for this file and, if present, skips the normal invocation of *fsck*(8).

SEE ALSO

halt(8), reboot(8), rc(8)

NAME

fsck – file system consistency check and interactive repair

SYNOPSIS

/etc/fsck **−p** [filesystem ...]
/etc/fsck [**−b** block#] [**−y**] [**−n**] [filesystem] ...

DESCRIPTION

The first form of *fsck* preens a standard set of filesystems or the specified file systems. It is normally used in the script **/etc/rc** during automatic reboot. In this case *fsck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of i/o overlap to check the file systems as quickly as possible. Normally, the root file system will be checked on pass 1, other "root" ("a" partition) file systems on pass 2, other small file systems on separate passes (e.g. the "d" file systems on pass 3 and the "e" file systems on pass 4), and finally the large user file systems on the last pass, e.g. pass 5. A pass number of 0 in fstab causes a disk to not be checked; similarly partitions which are not shown as to be mounted "rw" or "ro" are not checked.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the super-block wrong

These are the only inconsistencies which *fsck* with the **−p** option will correct; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot will then fail. For each corrected inconsistency one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. After successfully correcting a file system, *fsck* will print the number of files on that file system and the number of used and free blocks.

Without the **−p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that a number of the corrective actions which are not fixable under the **−p** option will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **−n** action.

Fsck has more consistency checks than its predecessors *check, dcheck, fcheck,* and *icheck* combined.

The following flags are interpreted by *fsck.*

−b Use the block specified immediately after the flag as the super block for the file system. Block 32 is always an alternate super block.

−y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

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

If no filesystems are given to *fsck* then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Directory size not of proper format.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:

   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster.

FILES

/etc/fstab	contains default list of file systems to check.

DIAGNOSTICS

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

SEE ALSO

fstab(5), fs(5), newfs(8), mkfs(8), crash(8S), reboot(8)

BUGS

Inode numbers for **.** and **..** in each directory should be checked for validity.

There should be some way to start a **fsck −p** at pass *n*.

NAME

fsirand – install random inode generation numbers

SYNOPSIS

fsirand [**–p**] special

DESCRIPTION

Fsirand installs random inode generation numbers on all the inodes on device *special,* and also installs a filesystem ID in the superblock. This helps increase the security of filesystems exported by NFS.

Fsirand must be used only on an unmounted filesystem that has been checked with *fsck*(8). The only exception is that it can be used on the root filesystem in single-user mode, if the system is immediately re-booted afterwords.

OPTIONS

–p Print out the generation numbers for all the inodes, but do not change the generation numbers.

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/usr/etc/in.ftpd host.socket

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Prototocol server process. The server is invoked by the Internet daemon *inetd*(8C) each time a connection to the ftp service (see *services*(5)) is made, with the connection available as descriptor 0 and the host and socket the connection originated from (in hex and decimal respectively) as argument.

Inactive connections are timed out after 60 seconds.

The ftp server currently supports the following ftp requests; case is not distinguished.

Request	**Description**
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -lg")
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PORT	specify data connection port
QUIT	terminate session
RETR	retrieve a file
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 765 are recognized, but not implemented.

Ftpd interprets file names according to the "globbing" conventions used by *csh*(1). This allows users to utilize the metacharacters "*?[]{}˜".

Ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */usr/etc/ftpusers*.

3) If the user name is "anonymous" or "ftp", an anonymous ftp account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended.

~ftp Make the home directory owned by "ftp" and unwritable by anyone.

~ftp/bin Make this directory owned by the super-user and unwritable by anyone. The program *ls*(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc Make this directory owned by the super-user and unwritable by anyone. The files *passwd*(5) and *group*(5) must be present for the *ls* command to work properly. These files should be mode 444.

~ftp/pub
Make this directory mode 777 and owned by "ftp". Users should then place files which are to be accessible via the anonymous account in this directory.

SEE ALSO

ftp(1C), ftpusers(5)

BUGS

There is no support for aborting commands.

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

gettable – get NIC format host tables from a host

SYNOPSIS

/etc/gettable *host*

DESCRIPTION

Gettable is a simple program used to obtain the NIC standard host tables from a "nicname" server. The indicated *host* is queried for the tables. The tables, if retrieved, are placed in the file *hosts.txt*.

Gettable operates by opening a TCP connection to the port indicated in the service specification for "nicname". A request is then made for "ALL" names and the resultant information is placed in the output file.

Gettable is best used in conjunction with the *htable*(8) program which converts the NIC standard file format to that used by the network library lookup routines.

SEE ALSO

intro(3N), htable(8)

BUGS

Should allow requests for only part of the database.

NAME

getty – set terminal mode

SYNOPSIS

/etc/getty [char]

DESCRIPTION

Getty is invoked by *init*(8) immediately after a terminal is opened, following the making of a connection. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the terminal line. Arguments other than '0' can be used to make *getty* treat the line specially. Refer to *ttys*(5) and *gettytab*(5) for descriptions of how *getty* uses the data in *gettytab* to set up the terminal. Normally, it sets the speed of the interface to 300 baud, specifies that raw mode is to be used (break on every character), that echo is to be suppressed, and either parity allowed. It types a banner identifying the system (from gethostname(2)) and the 'login:' message. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is then changed to 1200 baud and the 'login:' is typed again; a second 'break' changes the speed to 150 baud and the 'login:' is typed again. Successive 'break' characters cycle through the speeds 300, 1200, and 150 baud.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *tty*(4)).

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

Finally, *login* is called with the user's name as argument.

SEE ALSO

init(8), login(1), ioctl(2), tty(4), ttys(5), gettytab(5)

NAME

gxtest – stand alone test for the Sun video graphics board

SYNOPSIS

b /stand/gxtest

DESCRIPTON

Gxtest runs stand alone, not under control of the UNIX operating system. With the PROM resident monitor in control of the system, type the command:

```
> b /stand/gxtest
```

and the monitor boots the video test program into memory. *Gxtest* is completely self-explanatory and runs under its own steam. It reports any errors it finds on the screen.

NAME

halt – stop the processor

SYNOPSIS

/etc/halt [**–n**] [**–q**] [**–y**]

DESCRIPTION

Halt writes out sandbagged information to the disks and then stops the processor.

OPTIONS

–n Prevents the *sync* before stopping.

–q Do a quick halt, no graceful shutdown is attempted.

–y Needed if you are trying to halt the system from a dialup.

SEE ALSO

reboot(8), shutdown(8)

NAME

htable – convert NIC standard format host tables

SYNOPSIS

/usr/etc/htable *file*

DESCRIPTION

Htable is used to convert host files in the format specified in Internet RFC 810 to the format used by the network library routines. Three files are created as a result of running *htable*: *hosts*, *networks*, and *gateways*. The *hosts* file is used by the *gethostent*(3N) routines in mapping host names to addresses. The *networks* file is used by the *getnetent*(3N) routines in mapping network names to numbers. The *gateways* file is used by the routing daemon in identifying "passive" Internet gateways; see *routed*(8C) for an explanation.

If any of the files *localhosts*, *localnetworks*, or *localgateways* are present in the current directory, the file's contents is prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

Htable is best used in conjunction with the *gettable*(8C) program which retrieves the NIC database from a host.

SEE ALSO

intro(3N), gettable(8C)

BUGS

Does not properly calculate the *gateways* file.

NAME

icheck — file system storage consistency check

SYNOPSIS

/usr/etc/icheck [**−s**] [**−b** numbers] [filesystem]

DESCRIPTION

N.B.: *Icheck* is obsoleted for normal consistency checking by *fsck*(8).

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of *icheck* includes a report of

> The total number of files and the numbers of regular, directory, block special and character special files.
>
> The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.
>
> The number of free blocks.
>
> The number of blocks missing; i.e. not in any file nor in the free list.

The **−s** option causes *icheck* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **−s** option causes the normal output reports to be suppressed.

Following the **−b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), dcheck(8), ncheck(8), fs(5), clri(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. '*n* dups in free' means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since *icheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

NAME

ifconfig – configure network interface parameters

SYOPNSIS

/etc/ifconfig interface [*Ethernet_address*] [*hostname*] [*parameters*]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or to configure network interface parameters. *Ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address. Used without options, *ifconfig* displays the current configuration for a network interface. Only the super-user may modify the configuration of a network interface.

The **interface** parameter is a string of the form "name unit", for example "ie0".

OPTIONS

Ethernet_address is the hardware Ethernet address of a given machine. The address is a 6-byte hexadecimal value; each byte of the address is separated by a colon. A typical Ethernet adddress is '8:0:20:1:1:A3'. This address is contained in the ID PROM on the Sun-2 CPU Board, and is reported at boot time as one of the PROM monitor's sign-on messages. The *Ethernet_address* option is normally not used — the hardware supplies the default; the option is used only when trying to talk to a device which does not support ARP (like a VAX on a 4.1c network).

hostname may be either the hostname of a given machine (present in the hostname database, *hosts*(5)), or the complete Internet address consisting of your system's network number and the machine's unique host number. A typical Internet address might be "192.9.200.44", where "192.9.200" is the network number and "44" is the machine's hostnumber. To find a machine's Internet address, consult the local */etc/hosts* file.

The following *parameters* may be set with *ifconfig*:

up Mark an interface "up".

down Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface.

trailers Enable the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver.

—trailers Disable the use of a "trailer" link level encapsulation.

arp Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addreses and 10Mb/s Ethernet addresses.

—arp Disable the use of the Address Resolution Protocol.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

rc(8), intro(4N), netstat(1)

NAME

imemtest – stand alone memory test

SYNOPSIS

b /stand/imemtest

DESCRIPTON

Imemtest runs stand alone, not under control of the UNIX operating system. With the PROM resident monitor in control of the system, type the command:

```
> b /stand/imemtest
```

and the monitor boots the memory test program into memory. *Imemtest* is completely self-explanatory. It prompts for all start and end addresses, and after that it runs under its own steam. It reports any errors it finds on the screen.

NAME

inetd – internet services daemon

SYNOPSIS

/etc/inetd [**–d**]

DESCRIPTION

Inetd is the internet super-server which invokes all internet server processes as needed. Connection-oriented services are invoked each time a connection is made, by creating a process. This process is passed the connection as file descriptor 0 and an argument of the form "sourcehost.sourceport" where sourcehost is hex and sourceport is decimal.

Datagram oriented services are invoked when a datagram arrives; a process is created and passed the connection as file descriptor 0. Inetd will look at the socket where datagrams arrive again only after this process completes. The paradigms for such proceses are either to read off the incoming datagram and then fork and exit, or to process the arriving datagram and then time out.

Inetd consults *servers*(5) when it is invoked, and supports whatever services are in that file.

An rcp server can be started from inetd. The only differences from the usual code are that svcudp_create should be called as:

```
transp = scvudp_create(0)
```

since inet passes a socket file as descriptor 0, and svc_register should be called as:

```
svc_register(PROGNUM, VERSNUM, service, transp, 0)
```

with the final flag as 0, since the program will already have been registered by inetd. If you want to exit from the server process and return control to inet, you must explicitly exit since scv_run never returns.

The format of entries in */etc/servers* for rpc services is:

rcp udp *server-program program-number version-number*

where *server-program* is the C code implementing the server, and *program-number, version-number* are the program, and version numbers, respectively of the service. The keyword **udp** can be replaced by **tcp** for *tcp*-based services.

If the same program handles multiple versions, the version number can be specified as a range:

rcp udp /usr/etc/rstatd 100001 1-2

OPTIONS

–d Specifies that debugging traces are to be turned on for connection-oriented (TCP) services.

FILES

/etc/servers list of internet server processes

SEE ALSO

servers(5), comsat(8C), ftpd(8C), rexecd(8C), rlogind(8C), syslog(8), rshd(8C), talkd(8C), telnetd(8C), tftpd(8C)

BUGS

There is no provision for selectively invoking TCP debugging packet tracing per-service.

Should reread the */etc/servers* file on receipt of a SIGHUP signal. The */etc/servers* file can have no more than 26 lines.

NAME

init – process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

Init is invoked inside UNIX as the last step in the boot procedure. It normally then runs the automatic reboot sequence as described in *reboot*(8), and if this succeeds, begins multi-user operation. If the reboot fails, it commences single user operation by giving the super-user a shell on the console. It is possible to pass parameters from the boot program to *init* so that single user operation is commenced immediately. When such single user operation is terminated by killing the single-user shell (i.e. by hitting ^D), *init* runs */etc/rc* without the reboot parameter. This command file performs housekeeping operations such as removing temporary files, mounting file systems, and starting daemons.

In multi-user operation, *init's* role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file */etc/ttys* and forks several times to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input and output and the diagnostic output. Opening the terminal will usually involve a delay, since the *open* is not completed until someone is dialed up and carrier established on the channel. If a terminal exists but an error occurs when trying to open the terminal *init* complains by writing a message to the system console; the message is repeated every 10 minutes for each such terminal until the terminal is shut off in /etc/ttys and init notified (by a hangup, as described below), or the terminal becomes accessible (init checks again every minute). After an open succeeds, */etc/getty* is called with argument as specified by the second character of the *ttys* file line. *Getty* reads the user's name and invokes *login* to log in the user and execute the Shell.

Ultimately the Shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users, and makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. The *wtmp* entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and *getty* is reinvoked.

Init catches the *hangup* signal (signal SIGHUP) and interprets it to mean that the file */etc/ttys* should be read again. The Shell process on each line which used to be active in *ttys* but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add phone lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use 'kill –HUP 1.'

Init will terminate multi-user operations and resume single-user mode if sent a terminate (TERM) signal, i.e. "kill –TERM 1". If there are processes outstanding which are deadlocked (due to hardware or software failure), *init* will not wait for them all to die (which might take forever), but will time out after 30 seconds and print a warning message.

Init will cease creating new *getty*'s and allow the system to slowly die away, if it is sent a terminal stop (TSTP) signal, i.e. "kill –TSTP 1". A later hangup will resume full multi-user operations, or a terminate will initiate a single user shell. This hook is used by *reboot*(8) and *halt*(8).

Init's role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the *init* process cannot be located, the system will loop in user mode at location 0x13.

DIAGNOSTICS

init: *tty***: cannot open.** A terminal which is turned on in the *rc* file cannot be opened, likely because the requisite lines are either not configured into the system or the associated device was not attached during boot-time system configuration.

WARNING: Something is hung (wont die); ps axl advised. A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

FILES

/dev/console, /dev/tty*, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), getty(8), rc(8), reboot(8), halt(8), shutdown(8)

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [interval [count]]

DESCRIPTION

Iostat iteratively reports the number of characters read and written to terminals, and, for each disk, the number of seeks and transfers per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each fiftieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices it is possible to determine average seek times for each device.

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

The optional *count* argument restricts the number of reports.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(8)

NAME

kgmon – generate a dump of the operating system's profile buffers

SYNOPSIS

/usr/etc/kgmon [**–b**] [**–h**] [**–r**] [**–p**] [system] [memory]

DESCRIPTION

Kgmon is a tool used when profiling the operating system. When no arguments are supplied, *kgmon* indicates the state of operating system profiling as running, off, or not configured (see *config*(8)). If the **–p** flag is specified, *kgmon* extracts profile data from the operating system and produces a *gmon.out* file suitable for later analysis by *gprof*(1).

OPTIONS

–b Resume the collection of profile data.

–h Stop the collection of profile data.

–p Dump the contents of the profile buffers into a *gmon.out* file.

–r Reset all the profile buffers. If the **–p** flag is also specified, the *gmon.out* file is generated before the buffers are reset.

If neither **–b** nor **–h** is specified, the state of profiling collection remains unchanged. For example, if the **–p** flag is specified and profile data is being collected, profiling is momentarily suspended, the operating system profile buffers are dumped, and profiling is immediately resumed.

FILES

/vmunix – the default system
/dev/kmem – the default memory

SEE ALSO

gprof (1), config(8)

DIAGNOSTICS

Users with only read permission on /dev/kmem cannot change the state of profiling collection. They can get a *gmon.out* file with the warning that the data may be inconsistent if profiling is in progress.

NAME

lpc – line printer control program

SYNOPSIS

/usr/etc/lpc [*command* [*command-parameter* ...]]

DESCRIPTION

Lpc is the Line Printer Control program and is used by the system administrator to control the operation of the line printer system. For each line printer configured in */etc/printcap* (the line printer description database), *lpc* may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer daemons.

Without any arguments, *lpc* works interactively and prompts for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a *command* and the remaining arguments as *parameters* to the *command*. The standard input may be redirected so that *lpc* reads commands from a file. Commands may be abreviated; the following is the list of recognized commands. Note that the *printer* parameter in the commands is specified just by the name of the printer (**printronix** for example), not by **–Pprintronix** as you would specify it to the *lpr* or *lpq* commands.

? [command ...]

help [command ...]

Display a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort { all ¦ printer ... }

Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by *lpr*) for the specified printers. The **abort** command can only be used by the super-user.

clean { all ¦ printer ... }

Remove all files beginning with 'cf', 'tf', or 'df' from the specified printer queue(s) on the local machine. The **clean** command can only be used by the super-user.

enable { all ¦ printer ... }

Enable spooling on the local queue for the listed printers, so that *lpr* can put new jobs in the spool queue. The **enable** command can only be used by the super-user.

exit

quit

Exit from *lpc*.

disable { all ¦ printer ... }

Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*. The **disable** command can only be used by the super-user.

restart { all ¦ printer ... }

Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *Lpq* reports that there is no daemon present when this condition occurs.

start { all ¦ printer ... }

Enable printing and start a spooling daemon for the listed printers. The **start** command can only be used by the super-user.

status [all] [printer ...]

Display the status of daemons and queues on the local machine.

stop { all ¦ printer ... }

Stop a spooling daemon after the current job completes and disable printing. The **stop** command can only be used by the super-user.

topq printer [jobnum ...] [user ...]

Move the print job(s) specified by *jobnum* or those job(s) belonging to *user* to the top (head) of the printer queue. The **topq** command can only be used by the super-user.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/lock	lock file for queue control

SEE ALSO

lpd(8), lpr(1), lpq(1), lprm(1), printcap(5)

DIAGNOSTICS

?Ambiguous command	abreviation matches more than one command
?Invalid command	no match was found
?Privileged command	command can be executed by super-user only

NAME

lpd – line printer daemon

SYNOPSIS

/usr/lib/lpd [-l] [-L logfile] [port #]

DESCRIPTION

Lpd is the line printer daemon (spool area handler) and is normally invoked at boot time from the *rc*(8) file. It makes a single pass through the *printcap*(5) file to find out about the existing printers and prints any files left after a crash. It then uses the system calls *listen*(2) and *accept*(2) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is normally obtained with *getservent*(3N) but can be changed with the *port#* argument. The **–L** option changes the file used for writing error conditions from the system console to *logfile*. The **–l** flag causes *lpd* to log valid requests received from the network. This can be useful for debugging purposes.

Access control is provided by two means. First, all requests must come from one of the machines listed in the file */etc/hosts.equiv*. Second, if the "rs" capability is specified in the *printcap* entry for the printer being accessed, *lpr* requests will only be honored for those users with accounts on the machine with the printer.

The file *lock* in each spool directory is used to prevent multiple daemons from becoming active simultaneously, and to store information about the daemon process for *lpr*(1), *lpq*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files beginning with *cf*. Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

J Job Name. String to be used for the job name on the burst page.

C Classification. String to be used for the classification line on the burst page.

L Literal. The line contains identification info from the password file and causes the banner page to be printed.

T Title. String to be used as the title for *pr*(1).

H Host Name. Name of the machine where *lpr* was invoked.

P Person. Login name of the person who invoked *lpr*. This is used to verify ownership by *lprm*.

M Send mail to the specified user when the current print job completes.

f Formatted File. Name of a file to print which is already formatted.

l Like "f" but passes control characters and does not make page breaks.

p Name of a file to print using *pr*(1) as a filter.

t Troff File. The file contains *troff*(1) output (C/A/T phototypesetter commands).

d DVI File. The file contains output (DVI format from Stanford).

g Graph File. The file contains data produced by *plot*(3X).

c Cifplot File. The file contains data produced by *cifplot*.

v The file contains a raster image.

r The file contains text data with FORTRAN carriage control characters.

1 Troff Font R. Name of the font file to use instead of the default.

2 Troff Font I. Name of the font file to use instead of the default.

3 Troff Font B. Name of the font file to use instead of the default.

4 Troff Font S. Name of the font file to use instead of the default.

W Width. Changes the page width (in characters) used by *pr*(1) and the text filters.

I Indent. The number of characters to indent the output by (in ascii).

U Unlink. Name of file to remove upon completion of printing.

N File name. The name of the file which is being printed, or a blank for the standard input (when *lpr* is invoked in a pipeline).

If a file can not be opened, a message will be placed in the log file (normally the console). *Lpd* will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

Lpd uses *flock*(2) to provide exclusive access to the lock file and to prevent multiple deamons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1) and *lprm*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/dev/lp*	line printer devices
/etc/hosts.equiv	lists machine names allowed printer access

SEE ALSO

lpc(8), pac(8), lpr(1), lpq(1), lprm(1), printcap(5)
4.2BSD Line Printer Spooler Manual

NAME

makedbm – make a yellow pages dbm file

SYNOPSIS

makedbm [**–i** yp_input_file] [**–o** yp_output_name] [**–d** yp_domain_name] [**–m** yp_master_name] infile outfile
makedbm [**–u** dbmfilename]

DESCRIPTION

Makedbm takes *infile* and converts it to a pair of files in *dbm*(3X) format, namely *outfile*.pag and *outfile*.dir. Each line of the input file is converted to a single *dbm* record. All characters up to the first tab or space form the key, and the rest of the line is the data. If a line ends with \, then the data for that record is continued on to the next line. It is left for the clients of the yellow pages to interpret #; *makedbm* does not itself treat it as a comment character. *Infile* can be –, in which case standard input is read.

Makedbm is meant to be used in generating *dbm* files for the yellow pages, and it generates a special entry with the key YP_LAST_MODIFIED, which is the date of *infile* (or the current time, if *infile* is –).

OPTIONS

–i Create a special entry with the key YP_INPUT_FILE.

–o Create a special entry with the key YP_OUTPUT_NAME.

–d Create a special entry with the key YP_DOMAIN_NAME.

–m Create a special entry with the key YP_MASTER_NAME.

–u Undo a *dbm* file. That is, print out a *dbm* file one entry per line, with a single space separating keys from values.

EXAMPLE

It is easy to write shell scripts to convert standard files such as */etc/passwd* to the key value form used by *makedbm*. For example,

```
#!/bin/awk -f
BEGIN { FS = ":"; OFS = "\t"; }
{ print $1, $0 }
```

takes the */etc/passwd* file and converts it to a form that can be read by *makedbm* to make the yellow pages file *passwd.byname*. That is, the key is a username, and the value is the remaining line in the */etc/passwd* file.

SEE ALSO

dbm(3X), newpasswd(8)

NAME

MAKEDEV – make system special files

SYNOPSIS

/dev/MAKEDEV *device...*

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the */dev* directory, as this is the normal location of special files. Arguments to *MAKEDEV* are usually of the form *device-name*? where *device-name* is one of the supported devices listed in section 4 of the manual and '?' is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

std Create the *standard* devices for the system; for example, /dev/console, /dev/tty.

local Create those devices specific to the local site. This request runs the shell file */dev/MAKEDEV.local*. Site specific commands, such as those used to setup dialup lines as 'ttyd?' should be included in this file.

Since all devices are created using *mknod*(8), this shell script is useful only to the super-user.

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use **sh -x MAKEDEV** in case of trouble.

SEE ALSO

intro(4), config(8), mknod(8)

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 (that is, 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, upper- and lower-case letters, and '.' and '/'. 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 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 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 useful key bits in the result.

Makekey is intended for programs that perform encryption (for instance, *ed* and *crypt*(1)). Usually makekey's input and output will be pipes.

SEE ALSO

crypt(1), ed(1)

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special size [nsect] [ntrack] [blksize] [fragsize] [ncpg] [minfree] [rps] [nbpi]

DESCRIPTION

N.B.: file system are normally created with the *newfs*(8) command.

Mkfs constructs a file system by writing on the special file *special.* The numeric size specifies the number of sectors in the file system. *Mkfs* builds a file system with a root directory and a *lost+found* directory (see *fsck*(8)). The number of i-nodes is calculated as a function of the file system size. No boot program is initialized by *mkfs* (see *newfs*(8)).

OPTIONS

The optional arguments allow fine tune control over the parameters of the file system.

Nsect specify the number of sectors per track on the disk.

Ntrack
specify the number of tracks per cylinder on the disk. = # heads

Blksize
gives the primary block size for files on the file system. It must be a power of two, currently selected from 4096 or 8192.

Fragsize
gives the fragment size for files on the file system. The **fragsize** represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range 512 to 8192.

Ncpg specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32.

Minfree
specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is 10%.

rps If a disk does not revolve at 60 revolutions per second, this parameter may be specified.

nbpi Number of bytes for which one i-node block is allocated. This parameter is currently set at one i-node block for every 2048 bytes.

Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8), tunefs(8)

McKusick, Joy, Leffler; *A Fast File System for Unix, System Internals Manual for the Sun Workstation.*

NAME

mknod — build special file

SYNOPSIS

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

DESCRIPTION

Mknod makes a special file. The first argument is the *name* of the entry. The second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (for example, unit, drive, or line number).

Mknod is only for use by system configuration people. Normally you should use */dev/MAKEDEV* instead when making special files.

SEE ALSO

mknod(2), makedev(8)

NAME

mkproto – construct a prototype file system

SYNOPSIS

/usr/etc/mkproto special proto

DESCRIPTION

Mkproto is used to bootstrap a new file system. First a new file system is created using *newfs*(8). *Mkproto* is then used to copy files from the old file system into the new file system according to the directions found in the prototype file *proto*. The prototype file contains tokens separated by spaces or new lines. The first tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **–bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or – to specify set-user-id mode or not. The third is **g** or – for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see *chmod*(1).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkproto* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8)

BUGS

There should be some way to specify links.

There should be some way to specify bad blocks.

Mkproto can only be run on virgin file systems. It should be possible to copy files into existent file systems.

NAME

monitor – system PROM monitor and command interpreter

SYNOPSIS

Interrupt UNIX bootstrap to activate command interpreter as described below.

DESCRIPTION

The CPU board of the Sun workstation contains a set of PROMs called the *monitor*, which control the system during startup. They normally test the system then search for and attempt to boot UNIX. However if you interrupt the boot procedure, they enter interactive mode and accept commands interactively.

To enter the interactive mode, power down or reset the system, then interrupt the UNIX bootstrap. When the message:

```
autoboot in progress
```

appears, enter either:

L1 -a (press L1, then a) from the keyboard, or

BREAK (press the break key) from a terminal.

The monitor displays its prompt:

```
>
```

COMMANDS

The list of monitor commands is divided into two sections; miscellaneous commands, and commands to open and examine system memory resources. These are clustered at the end because they all use the same syntax.

The miscellaneous commands are:

b[*pathname*]
Boot. By itself, *b* searches for and boots UNIX; when followed by a *pathname*, it boots the specified program. *b?* lists the possible boot devices.

c[*addr*[Continue executing. With an address, it executes starting there; without an address, it executes starting at the current PC.

g[*addr*][*param*]
Start program execution. Starts executing code at the specified address; the default is the current PC.

k[*number*]
Reset. The default is 0 which resets the system without changing memory or the maps; 1 resets the system except for memory, and 2 resets the entire system including memory and the maps.

s[*number*]
Set or query the address space to be used by subsequent access commands. *number* represents the function code to be used; no number prints the current function code.

u[*dev*][*arg*]
Change UART assignment. This command manipulates the on-board UARTs and switches the current input and output device. It accepts the following devices:

a — SIO-A
b — SIO-B
s — screen
k — keyboard

And the following arguments:

i — input
o — output
(a number) — UART speed
u *addr* — Set virtual UART address.

The commands **a**, **d**, **e**, **l**, **m**, **o**, **p**, and **r** *open* various system resources so that you can examine and change the contents. These commands all use the following syntax:

c[*addr*][*a*...]

where:

c is the command

addr is the address (in hex) or the register number

a is a desired action. A hex number means "write this as data"; any non-hex character means "read this location"

The following paragraphs describe these commands:

a[*addr*][*a*...]
Open A-register. *n* is between 0 and 7, 0 starts at the boundary defined by the command *s*, and 7 is the stack pointer.

d[*addr*][*a*...]
Open D-register. *n* is 0 to 7, with default 0.

e[*addr*][*a*...]
Open word in memory. Default address is 0 as set by the command *s*, and odd addresses are rounded down.

l[*addr*][*a*...]
Open longword. Opens the longword at the address *addr*; the default is 0 in the address space set by *s*, and odd addresses are rounded down.

m[*addr*][*a*...]
Open segment map entry which maps virtual address *addr* in the current context. Current *s* command setting determines supervisor or user context. (0-3 = user; 4-7 = supervisor).

o[*addr*][*a*...]
Open byte location. Opens the byte at the specified location; the default is 0 in the address space specified by *s*.

p[*addr*][*a*...]
Open page map entry which maps virtual address *addr* in the current context. The default is 0. The choice of context is determined by the current *s* setting (0-3 = user; 4-7 = supervisor). Displays the revelant segment map entry with each page map entry in brackets.

r[*addr*][*a*...]
Open miscellaneous registers, as follows:

ss	is the supervisor stack pointer
us	is the user stack pointer
sf	is the source function code
df	is the destination function code
vb	is the vector base
sc	is the system context

uc is the user context

sr is the status register

pc is the program counter.

Changes made to these registers (except sc and vc) do not take effect until the next *c* command.

NAME

mount, umount − mount and dismount filesystems

SYNOPSIS

/etc/mount
/etc/mount −p
/etc/mount −a[**fv**][**t** type]
/etc/mount [**−frv**][**to** type options] [fsname] [dir]

/etc/umount [**−av**] [fsname ¦ dir] ...

DESCRIPTION

Mount announces to the system that a filesystem *fsname* is to be attached to the file tree at the directory *dir*. The directory *dir* must already exist. It becomes the name of the newly mounted root. The contents of *dir* are hidden until the filesystem is unmounted. If *fsname* is of the form host:path the filesystem type is assumed to be *nfs*(4).

Umount announces to the system that the filesystem *fsname* previously mounted on directory *dir* should be removed. Either the filesystem name or the mounted-on directory may be used.

Mount and *umount* maintain a table of mounted filesystems in */etc/mtab*, described in *mtab*(5). If invoked without an argument, *mount* displays the table. If invoked with only one of *fsname* or *dir* mount searches */etc/fstab* for an entry whose *dir* or *fsname* field matches the given argument. For example,

mount /usr

and

mount /dev/xy0g

are shorthand for

mount /dev/xy0g /usr

if this line is in */etc/fstab*

/dev/xy0g /usr 4.2 rw 1 1

MOUNT OPTIONS

−a Attempt to mount all the filesystems described in */etc/fstab*. In this case, *fsname* and *dir* are taken from */etc/fstab*. If a type is specified all of the filesystems in */etc/fstab* with that type will be mounted.

−o The next argument is a string that specifies mount options. Valid options are: ro, rw, quota, noquota, hard, soft. Hard and soft only make sense on *nfs*(4) filesystems. Options are separated by commas. The options *ro* and *rw* stand for read-only and read-write; *rw* is the default. Since quotas are not implemented, *noquota* is the default. With a hard remote mount, *mount* tries forever if the *mountd*(8c) server does not respond. Once the filesystem is mounted, access requests will retry forever if the *nfsd*(8) server does not respond. Hard is the default. With a soft remote mount, if the *mountd*(8c) server does not respond, *mount* forks a background copy to retry forever. Once the soft mount completes, access requests will fail with [ETIMEDOUT] if the *nfsd*(8) server does not respond.

−r Mount the specified filesystem read-only. This is a shorthand for:

mount −o ro fsname dir

Physically write-protected and magnetic tape filesystems must be mounted read-only, or errors will occur when access times are updated, whether or not any explicit write is attempted.

−t The next argument is the filesystem type. The accepted types are: 4.2, nfs, and pc; see *fstab*(5) for a description of the legal filesystem types.

−f Fake a new */etc/mtab* entry, but do not actually mount any filesystems.

−p Print the list of mounted filesystems in a format suitable for use in */etc/fstab*.

−v Verbose — *mount* displays a message indicating the filesystem being mounted.

UMOUNT OPTIONS

−a Attempt to unmount all the filesystems currently mounted. In this case, *fsname* is taken from */etc/mtab*.

−v Verbose — *umount* displays a message indicating the filesystem being unmounted.

EXAMPLES

mount /dev/xy0g /usr	mount a local disk
mount −ft 4.2 /dev/nd0 /	fake an entry for nd root
mount −at 4.2	mount all 4.2 filesystems
mount −t nfs serv:/usr/src /usr/src	mount remote filesystem
mount serv:/usr/src /usr/src	same as above
mount −o hard serv:/usr/src /usr/src	same as above but hard mount
mount −p > /etc/fstab	save current mount state

FILES

/etc/mtab	mount table
/etc/fstab	filesystem table

SEE ALSO

mount(2), nfsmount(2), unmount(2), fstab(5), mountd(8c), nfsd(8c)

BUGS

Mounting filesystems full of garbage will crash the system.

No more than one user should mount a disk partition "read-write" or the file system may become corrupted.

NAME

mountd – NFS mount request server

SYNOPSIS

/usr/etc/rpc.mountd

DESCRIPTION

Mountd is an *rpc*(4) server that answers file system mount requests. It reads the file */etc/exports*, described in *exports*(5), to determine which file systems are available to which machines and users. It also provides information as to which clients have file systems mounted. This information can be printed using the *showmount*(8) command.

The *mountd* daemon is normally invoked by *inetd*(8C).

SEE ALSO

nfs(4), rpc(4), exports(5), services(5), inetd(8), showmount(8)

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/usr/etc/ncheck [**–i** *numbers*] [**–a**] [**–s**] [*filesystem*]

DESCRIPTION

N.B.: For most normal file system maintenance, the function of *ncheck* is subsumed by *fsck*(8).

Ncheck with no argument generates a pathname versus i-number list of all files on a set of default file systems. Names of directory files are followed by '**/.**'.

A file system may be specified by the optional *filesystem* argument.

The report is in no useful order, and probably should be sorted.

OPTIONS

–i *numbers*
Report only those files whose i-*numbers* follow.

–a Print the names '**.**' and '**..**', which are ordinarily suppressed.

–s Report only special files and files with set-user-ID mode. This is intended to discover concealed violations of security policy.

SEE ALSO

sort(1), dcheck(8), fsck(8), icheck(8)

DIAGNOSTICS

When the filesystem structure is improper, '??' denotes the 'parent' of a parentless file and a pathname beginning with '**...**' denotes a loop.

NAME

nd — network disk control

SYNOPSIS

/etc/nd [*command*]

DESCRIPTION

The *nd* command controls the network disk service of the kernel as described in *nd*(4p). A single command may be given on the command line; if none is given then the standard input is read for a list of commands. Typically, the file /etc/nd.local is used for input. Lines beginning with '#' are considered to be comments.

The available commands are:

user *hostname hisunit mydev myoff mysize mylunit*

For the client *hostname* transform incoming requests for *hisunit* into server device *mydev* at offset *myoff* and size *mysize* sectors. /dev/ndl*mylunit* provides a local name for this disk "subpartition". If *mysize* is "-1", then this user unit is equivalent to the entire filesystem partition *mydev* (no "subpartioning" is done.) If *mylunit* is "-1" then no local name is needed for this user unit; this is usually the case with a swap unit, or a unit represented by an entire filesystem. If *hostname* is a numeric zero, *hisunit* refers to a public unit.

ether *hostname ether_addr [maxpacks]*

The **ether** command associates an Ethernet address *ether_addr* with the client *hostname*. This information is required in order for a client to learn his Internet address when it boots -- *nd* provides the Internet address for *hostname* based on the Ethernet address received in a boot request. The Ethernet address is given as size hex bytes separated by colons, e.g., 8:0:20:1:1:a3. At least one previous **user** command must have been issued for the client. The *maxpacks* option may be given to set the maximum number of packets that the server will send to the client before requesting an acknowledge. The default is 6; it should suffice unless the client Ethernet interface is very slow.

version *versionnumber*

The **version** command gives the level of configuration of the server. Occasionally the need arises to reorganize or reload the diskless partitions. Since the clients will rewrite locally cached blocks, they must be kept from writing their filesystems until they reboot. Before such a reorganization occurs, the system manager should warn diskless users to save files and halt their machines. Modification of the partitions should occur with the disk server off. After modification is complete, *versionnumber* should be incremented to force users to reboot.

son

Starts the network disk server. This command should be issued after all **user, ether,** and **version** commands.

soff

Stops the disk server until a subsequent **son** command.

clear

Stops the disk server and clears all **user** and **ether** information.

serverat *hostname*

Systems with disks may use the **serverat** command to specify a disk server if they wish to user a network disk in addition to their locally attached disk. Even then, this command is only necessary if they wish to use a public network disk or if they wish to change network disk servers.

FILES

/etc/nd.local

SEE ALSO

nd(4p)

BUGS

No sanity checking of disk partitions is done.

NAME

netstat – show network status

SYNOPSIS

netstat [**–m**¦**–i**¦ **–h**¦**–r**] [**–a**] [**–n**] [**–s**] [**–t**] [**–A**] [*interval*] [*system*] [*core*]

DESCRIPTION

Netstat symbolically displays the contents of various network-related data structures.

The arguments, *system* and *core* allow substitutes for the defaults '/vmunix' and '/dev/kmem'.

If an *interval* is specified, *netstat* continuously displays the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information that *netstat* presents. The display formats are controlled by the options listed below and are described there.

OPTIONS

Not all of the options listed here can be used in combination. Some of the options select the information to be displayed, and some of the options further qualify that specific display. *Netstat* checks its options in the order **–m** (memory management statistics), **–i** (interface statistics), **–h** (host table state), and **–r** (routing tables), and presents a display for only one of these options.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol. Other display formats are controlled by the options listed below.

–m Show statistics recorded by the memory management routines (the network manages a "private share" of memory)

–i Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown) The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

–h Show the state of the IMP host table. This does not work in an environment where the IMP host tables do not exist.

–r Show the routing tables. The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The *flags* field shows the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. The *refcnt* field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route then discard it. The *use* field provides a count of the number of packets sent using that route. The *interface* entry indicates the network interface utilized for the route.

Options listed below provide further qualification for the display formats listed above.

–A Show the address of any associated protocol control blocks; used for debugging.

–a Show the state of all sockets; normally sockets used by server processes are not shown

–n Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically).

–s Show per-protocol statistics. When used with the **–r** option, the **–s** option displays routing statistics.

–t Add timer information to the interface display.

FURTHER NOTES

Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **−n** option is specified, the address is printed in the Internet "dot format"; refer to *inet*(3N) for more information regarding this format. Unspecified, or "wildcard", addresses and ports appear as "*".

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted. Every 24th line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

iostat(8), vmstat(8), hosts(5), networks(5), protocols(5), services(5), trpt(8C)

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

The kernel's tables can change while *netstat* is examining them, creating incorrect or partial displays.

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

Newaliases rebuilds the random access data base for the mail aliases file */usr/lib/aliases*. It must be run each time */usr/lib/aliases* is changed in order for the change to take effect.

SEE ALSO

aliases(5), sendmail(8)

NAME

newfs − construct a new file system

SYNOPSIS

/etc/newfs [**−v**] [**−n**] [**mkfs-options**] **special** [**disk-type**]

DESCRIPTION

Newfs is a "friendly" front-end to the *mkfs*(8) program. On the VAX, *newfs* will look up the type of disk a file system is being created on in the disk description file */etc/disktab*; on the Sun the disk type is determined by reading the disk label. *Newfs* then calculates the appropriate parameters to use in calling *mkfs*, then builds the file system by forking *mkfs* and, if the file system is a root partition, installs the necessary bootstrap programs in the initial 16 sectors of the device.

OPTIONS

−n Do not install the bootstrap programs.

−v *newfs* prints out its actions, including the parameters passed to *mkfs*.

Options which may be used to override default parameters passed to *mkfs* are:

−s size The size of the file system in sectors.

−b block-size
The block size of the file system in bytes.

−f frag-size
The fragment size of the file system in bytes.

−t #tracks/cylinder

−c #cylinders/group
The number of cylinders per cylinder group in a file system. The default value used is 16.

−m free space %
The percentage of space reserved from normal users; the minimum free space threshhold. The default value used is 10%.

−r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

FILES

/etc/disktab	for disk geometry and file partition information (VAX only)
/etc/mkfs	to actually build the file system
/usr/mdec	for boot strapping programs

SEE ALSO

fs(5), fsck(8), tunefs(8)

The Sun UNIX File System in the Sun *System Internals Manual.*

NAME

nfsd, biod – NFS daemons

SYNOPSIS

/etc/nfsd [nservers]

/etc/biod [nservers]

DESCRIPTION

Nfsd starts the *nfs*(4) server daemons that handle client filesystem requests. *Nservers* is the number of file system request daemons to start. This number should be based on the load expected on this server; four is a good number. If *nservers* is not specified it defaults to one.

Biod starts *nservers* asynchronous block I/O daemons. This command is used on a NFS client to handle read-ahead and write-behind of buffer cached blocks. A good value for *nservers* is four; if not specified it defaults to one.

SEE ALSO

nfs(4), rpc(4), mountd(8c), exports(5)

NAME

nfsstat − Network File System statistics

SYNOPSIS

nfsstat [**−csnrdz**]

DESCRIPTION

Nfsstat displays statistical information about the Network File System (NFS), Remote Procedure Call (RPC), and Network Disk (ND) interfaces to the kernel. It can also be used to reinitialize this information. If no options are given the default is

```
nfsstat −csnr
```

That is, print everything except ND information, and reinitialize nothing.

OPTIONS

−c Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **−n** and **−r** options to print client NFS or client RPC information only.

−s Display server information. Works like the **−c** option above.

−n Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **−c** and **−s** options to print client or server NFS information only.

−r Display RPC information. Works like the **−n** option above.

−d Display Network Disk (ND) information.

−z Zero (reinitialize) statistics. Can be combined with any of the above options to zero particular sets of statistics after printing them. The user must have write permission on */dev/kmem* for this option to work.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory

SEE ALSO

nfs(4)

NAME

pac – printer/plotter accounting information

SYNOPSIS

/usr/etc/pac [**–P***printer*] [**–p***price*] [**–s**] [**–r**] [**–c**] [name ...]

DESCRIPTION

Pac reads the printer/plotter accounting files, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars. If any *names* are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

OPTIONS

–P*printer*

Do accounting for the named *printer.* Normally, accounting is done for the default printer (site dependent) or the value of the PRINTER environment variable is used.

–p*price*

Use the value *price* for the cost in dollars instead of the default value of 0.02.

–c Sorted the output by cost; usually the output is sorted alphabetically by name.

–r Reverse the sorting order.

–s Summarize the accounting information on the summary accounting file; this summary is necessary since on a busy system, the accounting file can grow by several lines per day.

FILES

/usr/adm/?acct	raw accounting files
/usr/adm/?_sum	summary accounting files

BUGS

The relationship between the computed price and reality is as yet unknown.

NAME

ping – network debugging

SYNOPSIS

/usr/etc/ping host [timeout]

DESCRIPTION

Ping repeatedly sends an icmp echo packet to *host* and reports whether or not a reply was received. It keeps trying until *timeout* seconds have elapsed, or an answer is received. The default timeout is 20 seconds. The *host* argument can be a name or an internet address.

SEE ALSO

icmp(4P)

NAME

portmap – DARPA port to RPC program number mapper

SYNOPSIS

/usr/etc/rpc.portmap

DESCRIPTION

Portmap is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell *portmap* what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact *portmap* on the server machine to determine the port number where RPC packets should be sent.

Normally, standard RPC servers are started by *inetd*(8c), so *portmap* must be started before *inetd* is invoked.

SEE ALSO

servers(5), rpcinfo(8), inetd(8)

BUGS

If *portmap* crashes, all servers must be restarted.

NAME

pstat − print system facts

SYNOPSIS

/etc/pstat −alxptufT [*suboptions* [[*system* [*corefile*]]

DESCRIPTION

Pstat interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in */dev/kmem*. The required namelist is taken from */vmunix* unless *system* is specified.

OPTIONS

−a Under **−p**, describe all process slots rather than just active ones.

−i Print the inode table including the associated vnode entries with these headings:

LOC The core location of this table entry.

IFLAG Miscellaneous inode state variables encoded thus:

A	inode access time must be corrected
C	inode has been changed
L	inode is locked
R	inode is being referenced
T	inode contains a pure text prototype
U	update time (*fs*(5)) must be corrected
W	wanted by another process (L flag is on)
Z	someone waiting for an exclusive lock

IDEVICE Major and minor device number of file system in which this inode resides.

INO I-number within the device.

MODE Mode bits in octal, see *chmod*(2).

NLK Number of links to this inode.

UID User ID of owner.

SIZE/DEV Number of bytes in an ordinary file, or major and minor device of special file.

VFLAG Miscellaneous vnode state variables encoded thus:

R	root of its file system
S	shared lock applied
E	exclusive lock applied
T	vnode is a pure text prototype
Z	proccess is waiting for a shared or exclusive lock

CNT Number of open file table entries for this vnode.

SHC Reference count of shared locks on the vnode.

EXC Reference count of exclusive locks on the vnode (this may be > 1 if, for example, a file descriptor is inherited across a fork).

TYPE Vnode file type, either VNON (no type), VREG (regular), VDIR (directory), VBLK (block device), VCHR (character device), VLNK (symbolic link), VSOC (socket), or VBAD (bad).

−x Print the text table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

P	resulted from demand-page-from-inode exec format (see *execve*(2))
T	*ptrace*(2) in effect
W	text not yet written on swap device
L	loading in progress
K	locked
w	wanted (L flag is on)

DADDR Disk address in swap, measured in multiples of 512 bytes.

CADDR Head of a linked list of loaded processes using this text segment.

RSS Resident set size, measured in multiples of 512 bytes.

SIZE Size of text segment, measured in multiples of 512 bytes.

VPTR Core location of corresponding vnode.

CNT Number of processes using this text segment.

CCNT Number of processes in core using this text segment.

−p Print process table for active processes with these headings:

LOC The core location of this table entry.
S Run state encoded thus:
0 no process
1 awaiting an event
2 (abandoned state)
3 runnable
4 being created
5 being terminated
6 stopped under trace
F Miscellaneous state variables, or-ed together (hexadecimal):
0000001 loaded
0000002 a system process (scheduler or page-out daemon)
0000004 locked for swap out
0000008 swapped out during process creation
0000010 traced
0000020 used in tracing
0000040 user settable lock in core
0000080 in page-wait
0000100 prevented from swapping during *fork*(2)
0000200 restore old sigmask after taking signal
0000400 exiting
0000800 doing physical i/o
0001000 process resulted from a *vfork*(2) which is not yet complete
0002000 another flag for *vfork*(2)
0004000 process has no virtual memory, as it is a parent in the context of *vfork*(2)
0008000 process is demand paging data pages from its text inode
0010000 process has advised of sequential behavior with *vadvise*(2)
0020000 process has advised of anomalous behavior with *vadvise*(2)
0040000 process is in a sleep which will timeout
0080000 (unused)
0100000 using old signal mechanism
0200000 process is owed a profiling tick
0400000 process is setting up a *select*(2) call
0800000 Process is a login process
1000000 Page tables for this process have changed (tlb is dirty)
POIP number of pages currently being pushed out from this process.
PRI Scheduling priority, see *setpriority*(2).
SIG Signals received (signals 1-32 coded in bits 0-31),
UID Real user ID.
SLP Amount of time process has been blocked.

TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see *setpriority*(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
ADDR If in core, the page frame number of the first page of the 'u-area' of the process. If swapped out, the position in the swap area measured in multiples of 512 bytes.
RSS Resident set size – the number of physical page frames allocated to this process.
SRSS RSS at last swap (0 if never swapped).
SIZE Virtual size of process image (data+stack) in multiples of 512 bytes.
WCHAN Wait channel number of a waiting process.
LINK Link pointer in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for real interval timer, *setitimer*(2) measured in clock ticks (10 milliseconds).

−t Print table for terminals with these headings:

RAW Number of characters in raw input queue.
CAN Number of characters in canonicalized input queue.
OUT Number of characters in putput queue.
MODE See *tty*(4).
ADDR Physical device address.
DEL Number of delimiters (newlines) in canonicalized input queue.
COL Calculated column position of terminal.
STATE Miscellaneous state variables encoded thus:

T delay timout in progress
W waiting for open to complete
O open
C carrier is on
B busy doing output
A process is awaiting output
X open for exclusive use
H hangup on close

PGRP Process group for which this is controlling terminal.
DISC Line discipline; blank is old tty OTTYDISC or "new tty" for NTTYDISC or "net" for NETLDISC (see *bk*(4)).

−u print information about a user process; the next argument is its address as given by *ps*(1). The process must be in main memory, or the file used can be a core image and the address 0.

−f Print the open file table with these headings:

LOC The core location of this table entry.

TYPE The type of object the file table entry points to.
FLG Miscellaneous state variables encoded thus:

R open for reading
W open for writing
A open for appending
S shared lock present
X exclusive lock present

	I signal pgrp when data ready
CNT	Number of processes that know this open file.
MSG	Number of references from message queue.
DATA	The location of the vnode table entry or socket for this file.
OFFSET	The file offset (see *lseek*(2)), or the core address of the associated socket structure.

−s print information about swap space usage: the number of (512 byte) blocks used and free is given as well as the number of used blocks which belong to text images.

−T prints the number of used and free slots in the several system tables and is useful for checking to see how full system tables have become if the system is under heavy load.

FILES

/vmunix	namelist
/dev/kmem	default source of tables

SEE ALSO

ps(1), stat(2), fs(5)
K. Thompson, *UNIX Implementation*

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by **−T**; even more useful if these tables were dynamically allocated.

NAME

quot – summarize file system ownership

SYNOPSIS

/usr/etc/quot [**−cfhnv**] [*filesystem*]

DESCRIPTION

Quot displays the number of blocks (1024 bytes) in the named *filesystem* currently owned by each user. If no *filesystem* is named, a default name is used.

OPTIONS

−c Display three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

−f Display count of number of files as well as space owned by each user.

−h Estimate the number of blocks in the file — this doesn't account for files with holes in them.

−n Run the pipeline **ncheck filesystem ¦ sort +0n ¦ quot −n filesystem** to produce a list of all files and their owners.

−v Display three columns containing the number of blocks not accessed in the last 30, 60, and 90 days.

FILES

Default file system varies with system.
/etc/passwd to get user names

SEE ALSO

ls(1), du(1)

NAME

rc – command script for auto-reboot and daemons

SYNOPSIS

/etc/rc
/etc/rc.local

DESCRIPTION

Rc is the command script which controls the automatic reboot and *rc.local* is the script holding commands which are pertinent only to a specific site.

When an automatic reboot is in progress, *rc* is invoked with the argument *autoboot* and runs a *fsck* with option **–p** to "preen" all the disks of minor inconsistencies resulting from the last system shutdown and to check for serious inconsistencies caused by hardware or software failure. If this auto-check and repair succeeds, then the second part of *rc* is run.

The second part of *rc*, which is run after a auto-reboot succeeds and also if *rc* is invoked when a single user shell terminates (see *init*(8)), starts all the daemons on the system, preserves editor files and clears the scratch directory **/tmp.** *Rc.local* is executed immediately before any other commands after a successful *fsck*. Normally, the first commands placed in the *rc.local* file define the machine's name, using *hostname*(1), and save any possible core image that might have been generated as a result of a system crash, *savecore*(8). The latter command is included in the *rc.local* file because the directory in which core dumps are saved is usually site specific.

SEE ALSO

init(8), reboot(8), savecore(8)

NAME

rdate – set system date from a remote host

SYNOPSIS

/usr/ucb/rdate hostname

DESCRIPTION

Rdate sets the local date and time from the *hostname* given as argument. You must be superuser on the local system. Typically *rdate* can be inserted as part of your */etc/rc.local* startup script.

SEE ALSO

timed(8C)

BUGS

Could be modified to accept a list of hostnames and try each until a valid date returned. Better yet would be to write a real date server that accepted broadcast requests.

NAME

reboot – UNIX bootstrapping procedures

SYNOPSIS

/etc/reboot [**–n**] [**–q**]

DESCRIPTION

The UNIX operating system is started by placing it in memory transferring to it. Since the system is not re-enterable, it is necessary to read it in from disk or tape each time it is to be bootstrapped.

Rebooting a running system. When a UNIX system is running and a reboot is desired, *shutdown*(8) is normally used. If there are no users then **/etc/reboot** can be used. Reboot performs a *sync* operation on the disks, and then a multi-user reboot (as described below) is initiated. This causes a system to be booted and an automatic disk check to be performed. If all this succeeds without incident, the system is then brought up multi-user.

OPTIONS

–n option avoids the sync. It can be used if a disk or the processor is on fire.

–q reboots quickly and ungracefully, without first shutting down running processes.

Power fail and crash recovery. Normally, the system will reboot itself at power-up or after crashes. When it reboots, an automatic consistency check of the file systems is done. If this check is successful, the system resumes multi-user operations.

The *boot* program finds the corresponding file on the given device, loads that file into memory location zero, and starts the program at the entry address specified in the program header. Normal line editing characters can be used in specifying the pathname.

For tapes, the minor device number gives a file offset.

FILES

/vmunix	system code
/boot	system bootstrap

SEE ALSO

crash(8S), fsck(8), init(8), rc(8), shutdown(8), halt(8), newfs(8)

NAME

recnews — receive unprocessed articles via mail

SYNOPSIS

/usr/lib/news/recnews [*newsgroup* [*sender*]]

DESCRIPTION

Recnews reads a letter from the standard input; determines the article title, sender, and newsgroup; and gives the body to inews with the right arguments for insertion.

If *newsgroup* is omitted, the to line of the letter is used. If *sender* is omitted, the sender is determined from the from line of the letter. The title is determined from the subject line.

SEE ALSO

inews(1), uurec(8), sendnews(8), readnews(1), checknews(1)

NAME

renice – alter priority of running processes

SYNOPSIS

/etc/renice [**–g**] [**–u**] priority who ...

DESCRIPTION

Renice can be used to alter the scheduling priority of one or more running processes. By default, the processes to be affected are specified by their process id's. If the **–g** option is specified, the *who* parameters are interpreted as process groups and all the processes in the specified process groups have their scheduling priority altered. If the **–u** option is indicated, the *who* parameters are interpreted as user names and all process owned by the user are affected.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to PRIO_MIN (20). (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range PRIO_MAX (–20) to PRIO_MIN. Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to), 0 (the "base" scheduling priority), anything negative (to make things go very fast).

If no *who* parameter is specified, the current process (alternatively, process group or user) is used.

FILES

/etc/passwd	to map user names to user id's

SEE ALSO

getpriority(2)

BUGS

If you make the priority very negative, then the process cannot be interrupted. To regain control you must make the priority greater than zero. Non super-users can not increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

NAME

restore — incremental file system restore

SYNOPSIS

/etc/restore key [name ...]

DESCRIPTION

Restore restores files from tapes previously created via the *dump*(8) command. *Restore*'s 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 *restore* are file or directory names specifying the files that are to be restored. Unless the **h** key is specified (see below), the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

FUNCTION LETTERS

The function portion of the key is specified by one of the following letters:

r The tape is read and loaded into the current directory. This should not be done lightly; the **r** key should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape after a full level zero restore. Thus:

```
tutorial% /etc/newfs /dev/rxy0g eagle
tutorial% /etc/mount /dev/xy0g /mnt
tutorial% cd /mnt
tutorial% restore r
```

is a typical sequence to restore a complete dump. Another *restore* can be done to get an incremental dump in on top of this. A *dump*(8) followed by a *newfs*(8) and a *restore* can be used to change the size of a file system.

R *Restore* requests a particular tape of a multi volume set on which to restart a full restore (see the **r** key above). This allows *restore* to be interrupted and then restarted.

x Extract the named files from the tape. If the named file matches a directory whose contents had been written onto the tape, and the **h** key is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the root directory is extracted, which results in the entire content of the tape being extracted, unless the **h** key has been specified.

t List the names of the specified files if they occur on the tape. If no file argument is given, the root directory is listed, which results in the entire content of the tape being listed, unless the **h** key has been specified. Note that the **t** key replaces the function of the old *dumpdir* program.

i Restore files interactively from a dump tape. After reading in the directory information from the tape, *restore* provides a shell like interface within which the user can move around the directory tree selecting files to be extracted. The available commands that this 'shell' provides are given below. For those commands that require an argument, the default is the current directory.

ls [arg] List the current or specified directory. Entries that are directories are appended with a '/'. Entries that have been marked for extraction are prepended with a '*'. If the verbose key is set the inode number of each entry is also listed.

cd arg Change the current working directory to the specified argument.

pwd Print the full pathname of the current working directory.

add [arg] Add the current directory or specified argument to the list of files to be extracted. If a directory is specified, add that directory and all its to the extraction list (unless the **h** key is specified on the command line). Files that are on the extraction list are prepended with a '*' when they are listed by **ls**.

delete [arg]
Delete the current directory or specified argument from the list of files to be extracted. If a directory is specified, delete that directory and all its descendents from the extraction list (unless the **h** key is specified on the command line). The most expedient way to extract most of the files from a directory is to add the directory to the extraction list and then delete those files that are not needed.

extract Extract from the dump tape all the files that are on the extraction list. *Restore* asks which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

verbose Toggle the sense of the **v** key. When the verbose key is set, the **ls** command lists the inode numbers of all entries, and *restore* also displays information about each file as it is extracted.

help List a summary of the available commands.

quit Restore immediately exits, even if the extraction list is not empty.

FUNCTION MODIFIERS

The function modifiers which may be used in addition to the function letters are as follows. A few of these modifiers — namely **b**, **f**, and **s**— take an argument from the command line. If you use more than one of **b**, **f**, and **s**, nest your arguments as you do your modifiers; if you use **b** first on the command line, place its argument first on the command line, and so on.

b Specifies the blocking factor for the *restore*. The blocking factor is taken from the next argument on the command line. This corresponds to the **b** key in *dump*(8).

d Turns on debugging output.

v This is the verbose option. It means display the name of each file it treats preceded by its file type. *Restore* normally works silently.

f Use the next argument to *restore* as the name of the archive instead of /dev/rmt?. If the name of the file is '–', *restore* reads from standard input. Thus, *dump*(8) and *restore* can be used in a pipeline to dump and restore a file system with the command

```
tutorial% dump 0f – /usr ¦ (cd /mnt; restore xf –)
```

If the name of the file is 'machine:device' the restore is done from the specified machine through the internet using *rmt*(8C).

s The next argument to *restore* is the number of files to skip in the case where there are multiple dump files on the dump tape. For example, a command like

```
tutorial% restore xfs /dev/nrar0 5
```

would position you at the sixth file on the tape.

y Do not ask whether to abort the restore in the event of tape errors. *Restore* always tries to skip over the bad tape block(s) and continue as best it can.

m Extract by inode numbers rather than by file name. This is useful if only a few files are being extracted, and one wants to avoid regenerating the complete pathname to the file.

h Extract the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

c Convert the contents of the dump tape to the new file system format.

DIAGNOSTICS

Complaints about bad key characters.

Complaints if it gets a read error. If **y** has been specified, or the user responds 'y', *restore* will attempt to continue the restore.

If the dump extends over more than one tape, *restore* asks the user to change tapes. If the **x** or **i** key has been specified, *restore* also asks which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

There are numerous consistency checks that can be listed by *restore*. Most checks are self-explanatory or can 'never happen'. Common errors are given below.

Converting to new file system format.

> A dump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

<filename>: not found on tape

> The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file <inumber>, got <inumber>

> A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low

> When doing incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high

> When doing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or that has too high an incremental level has been loaded.

Tape read error while restoring <filename>
Tape read error while skipping over inode <inumber>
Tape read error while trying to resynchronize

> A tape read error has occurred. If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped <num> blocks

> After a tape read error, *restore* may have to resynchronize itself. This message lists the number of blocks that were skipped over.

FILES

/dev/rmt?	the default tape drive
/tmp/rstdir*	file containing directories on the tape.
/tmp/rstmode*	owner, mode, and time stamps for directories.
./restoresymtab	information passed between incremental restores.

SEE ALSO

dump(8), newfs(8), mount(8), mkfs(8), rmt(8C)

BUGS

Restore can get confused when doing incremental restores from dump tapes that were made on active file systems.

A level zero dump must be done after a full restore. Because restore runs in user mode, it has no control over inode allocation; this means that *restore* re-positions the files, although it does change their contents. Thus, a full dump must be done to get a new set of directories reflecting the new file positions, so that later incremental dumps will be correct.

NAME

rexecd – remote execution server

SYNOPSIS

/usr/etc/in.rexecd host.port

DESCRIPTION

Rexecd is the server for the *rexec*(3N) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords. It is invoked automatically as needed by *inetd*(8C), and then executes the following protocol:

1) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *Rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd.*

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong password was supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

SEE ALSO

inetd(8C)

NAME

rlogind — remote login server

SYNOPSIS

/etc/in.rlogind host.port

DESCRIPTION

Rlogind is the server for the *rlogin*(1C) program. The server provides a remote login facility with authentication based on privileged port numbers.

Rlogind is invoked by *inetd*(8C) when a remote login connection is established, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The client's address and port number are passed as arguments to *rlogind* by *inetd* in the form "host.port" with host in hex and port in decimal.

2) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(4)), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the **stdin** , **stdout** , and **stderr** for a login process. The login process is an instance of the *login*(1) program, invoked with the **−r** option. The login process then proceeds with the authentication process as described in *rshd*(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseudo terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty*(4) is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, "TERM"; see *environ*(5).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

SEE ALSO

inetd(8C)

NAME

rmail – handle remote mail received via uucp

SYNOPSIS

rmail user ...

DESCRIPTION

Rmail interprets incoming mail received via *uucp*(1C), collapsing "From" lines in the form generated by *binmail*(1) into a single line of the form "return-path!sender", and passing the processed mail on to *sendmail*(8).

Rmail is explicitly designed for use with *uucp* and *sendmail.*

SEE ALSO

binmail(1), uucp(1C), sendmail(8)

BUGS

Rmail should not reside in /bin.

NAME

rmt – remote magtape protocol module

SYNOPSIS

/etc/rmt

DESCRIPTION

Rmt is a program used by the remote dump and restor programs in manipulating a magnetic tape drive through an interprocess communication connection. *Rmt* is normally started up with an *rexec*(3N) or *rcmd*(3N) call.

The *rmt* program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

> **A***number*\n

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

> **E***error-number*\n*error-message*\n,

where *error-number* is one of the possible error numbers described in *intro*(2) and *error-message* is the corresponding error string as printed from a call to *perror*(3). The protocol is comprised of the following commands (a space is present between each token).

O device mode
: Open the specified *device* using the indicated *mode*. *Device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to *open*(2). If a device had already been opened, it is closed before a new open is performed.

C device
: Close the currently open device. The *device* specified is ignored.

L whence offset
: Perform an *lseek*(2) operation using the specified parameters. The response value is that returned from the *lseek* call.

W count
: Write data onto the open device. *Rmt* reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. The response value is that returned from the *write*(2) call.

R count
: Read *count* bytes of data from the open device. *Rmt* performs the requested *read*(2) and responds with **A***count-read*\n if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I operation count
: Perform a MTIOCOP *ioctl*(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the *ioctl* call. The return value is the *count* parameter when the operation is successful.

S
: Return the status of the open device, as obtained with a MTIOCGET *ioctl* call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

Any other command causes *rmt* to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

rcmd(3N), rexec(3N), mtio(4), dump(8), restore(8)

BUGS

People tempted to use this for a remote file access protocol are discouraged.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/usr/etc/route [**–f**] [*command args*]

DESCRIPTION

Route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *routed*(8C), should tend to this task.

Route accepts three commands: *add*, to add a route; *delete*, to delete a route; and *change*, to modify an existing route.

All commands have the following syntax:

> **/usr/etc/route** *command* **destination gateway** [**metric**]

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a "local address part" of INADDR_ANY, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database, *hosts*(5). If this lookup fails, the name is then looked for in the network name database, *networks*(5).

Route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **–f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

DIAGNOSTICS

"add %s: gateway %s flags %x"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

"delete %s: gateway %s flags %x"
As above, but when deleting an entry.

"%s %s done"
When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

"not in table"
A delete operation was attempted for an entry which wasn't present in the tables.

"routing table overflow"
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO

routing(4N), routed(8C)

BUGS

The change operation is not implemented, one should add the new route, then delete the old one.

NAME

routed – network routing daemon

SYNOPSIS

/etc/in.routed [**–s**] [**–q**] [**–t**] [*logfile*]

DESCRIPTION

Routed is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation *routed* listens on *udp*(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. *Routed* then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Request packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (that is, the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. *Routed* waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

Supplying the **–s** option forces *routed* to supply routing information whether it is acting as an internetwork router or not. The **–q** option is the opposite of the **–s** option. If the **–t** option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process. Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of "distant" *passive* and *active* gateways. When *routed* is started up, it reads the file */etc/gateways* to find gateways which may not be identified using the SIOGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (that is, they should have a *routed* process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The */etc/gateways* is comprised of a series of lines, each in the following format:

< **net** ¦ **host** > *name1* **gateway** *name2* **metric** *value* < **passive** ¦ **active** >

The **net** or **host** keyword indicates if the route is to a network or specific host.

Name1 is the name of the destination network or host. This may be a symbolic name located in */etc/networks* or */etc/hosts*, or an Internet address specified in "dot" notation; see *inet*(3N).

Name2 is the name or address of the gateway to which messages should be forwarded.

Value is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as *passive* or *active* (as described above).

FILES

/etc/gateways for distant gateways

SEE ALSO

Internet Transport Protocols, XSIS 028112, Xerox System Integration Standard. (Sun 800-1066-01)
udp(4P)

BUGS

The kernel's routing tables may not correspond to those of *routed* for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

Routed should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME

rpcinfo — report RPC information

SYNOPSIS

rpcinfo −p [host]
rpcinfo −u host program-number version-number
rpcinfo −t host program-number version-number

DESCRIPTION

Rpcinfo makes an RPC call to an RPC server and reports what it finds.

OPTIONS

−p Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the value returned by *hostname*(1).

−u Make an RPC call to procedure 0 of *program-number* using UDP, and report whether a response was received.

−t Make an RPC call to procedure 0 of *program-number* using TCP, and report whether a response was received.

SEE ALSO

RPC Reference Manual, portmap(8)

NAME

rshd – remote shell server

SYNOPSIS

/etc/in.rshd host.port

DESCRIPTION

Rshd is the server for the *rcmd*(3N) routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

Rshd is invoked by *inetd*(8C) each time a shell service is requested, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The clients host address (in hex) and port number (in decimal) are the argument passed to *rshd.*

2) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) *Rshd* then validates the user according to the following steps. The remote user name is looked up in the password file and a *chdir* is performed to the user's home directory. If the lookup or fails, the connection is terminated. If the *chdir* fails, it does a *chdir to /* (root). If the user is not the super-user, (user id 0), the file */etc/hosts.equiv* is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file *.rhosts* in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd.*

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Login incorrect."
No password file entry for the user name existed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the **stderr**, wasn't created.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

SEE ALSO

rsh(1C), rcmd(3N)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rstatd − kernel statistics server

SYNOPSIS

/usr/etc/rpc.rstatd

DESCRIPTION

Rstatd is a server which returns performance statistics obtained from the kernel. These statistics are graphically displayed by *perfmeter*(1). The *rstatd* daemon is normally invoked by *inetd*(8C).

SEE ALSO

perfmeter(1), services(5), inetd(8)

NAME

rwalld — network rwall server

SYNOPSIS

/usr/etc/rpc.rwalld

DESCRIPTION

Rwalld is a server that handles *rwall*(1) and *shutdown*(1) requests. It is implemented by calling *wall*(1) to all the appropriate network machines. The *rwalld* daemon is normally invoked by *inetd*(8C).

SEE ALSO

rwall(1), wall(1), services(5), inetd(8), shutdown(8)

NAME

rwhod — system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

Rwhod is the server which maintains the database used by the *rwho*(1C) and *ruptime*(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

Rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory */usr/spool/rwho*.

The *rwho* server transmits and receives messages at the port indicated in the "rwho" service specification, see *services*(5). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];      /* tty name */
        char    out_name[8];      /* user id */
        long    out_time;         /* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct          outmp we_utmp;
                int             we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the *w*(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the *gethostname*(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(5) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at a *rwho* server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named *whod.hostname* in the directory */usr/spool/rwho*. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. *Rwhod* performs an *nlist*(3) on /vmunix every 10 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1C), ruptime(1C)

BUGS

Should relay status information between networks. People often interpret the server dying as a machine going down.

NAME

sa, accton — system accounting

SYNOPSIS

/usr/etc/sa [**−abcdDfijkKlnrstuv**] [file]

/usr/etc/accton [file]

DESCRIPTION

With an argument naming an existing *file, accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

Sa reports on, cleans up, and generally maintains accounting files.

Sa is able to condense the information in */usr/adm/acct* into a summary file */usr/adm/savacct* which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system */usr/adm/acct* can grow by 500K bytes per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; */usr/adm/acct* is the default.

Output fields are labelled: 'cpu' for the sum of user+system time (in minutes), 're' for real time (also in minutes), 'k' for cpu-time averaged core usage (in 1k units), 'avio' for average number of I/O operations per execution. With options fields labelled 'tio' for total I/O operations, 'k*sec' for cpu storage integral (kilo-core seconds), 'u' and 's' for user and system cpu time alone (both in minutes) will sometimes appear.

OPTIONS

a Place all command names containing unprintable characters and those used only once under the name '***other.'

b Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

c Besides total user, system, and real time for each command print percentage of total time over all commands.

d Sort by average number of disk I/O operations.

D Print and sort by total number of disk I/O operations.

f Force no interactive threshold compression with −v flag.

i Don't read in summary file.

j Instead of total minutes time for each category, give seconds per call.

k Sort by cpu-time average memory usage.

K Print and sort by cpu-storage integral.

l Separate system and user time; normally they are combined.

m Print number of processes and number of CPU minutes for each user.

n Sort by number of calls.

r Reverse order of sort.

s Merge accounting file into summary file */usr/adm/savacct* when done.

t For each command report ratio of real time to the sum of user and system times.

u Superseding all other flags, print for each record in the accounting file the user ID and command name.

v Followed by a number *n*, types the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with 'y', add the command to the category '**junk**.' This is used to strip out garbage.

FILES

/usr/adm/acct raw accounting

SEE ALSO

ac(8), acct(2), acct(5)

NAME

savecore – save a core dump of the operating system

SYNOPSIS

/usr/etc/savecore *dirname* [*system*]

DESCRIPTION

Savecore is meant to be called near the end of the */etc/rc* file after the system boots. *Savecore*'s function is to save the core dump of the system (assuming one was made) and to write a reboot message in the shutdown log.

Savecore checks the core dump to be certain it corresponds with the current running version of the operating system. If it does, *savecore* saves the core image in the file *dirname*/vmcore.n and its brother, the namelist, *dirname*/vmunix.n The trailing *.n* in the pathnames is replaced by a number which grows every time *savecore* is run in that directory.

Before *savecore* writes out a core image, it reads a number from the file *dirname*/minfree. If there less free space on the filesystem which contains *dirname* than the number obtained from the minfree file, the core dump is not done. If the *minfree* file does not exist, *savecore* always writes out the core file (assuming that a core dump was taken).

Savecore also writes a reboot message in the shut down log. If the system crashed as a result of a panic, *savecore* records the panic string in the shut down log too.

If the core dump was from a system other than /vmunix, the name of that system must be supplied as *sysname*.

FILES

/usr/adm/shutdownlog	shut down log
/usr/crash/bounds	number to assign to the core images
/vmunix	current UNIX

BUGS

Can be fooled into thinking a core dump is the wrong size.

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [**–ba**] [**–bd**] [**–bi**] [**–bm**] [**–bp**] [**–bs**] [**–bt**] [**–bv**]
[**–C***file*] [**–d***X*] [**–F***fullname*] [**–f***name*] [**–h***N*] [**–n**] [**–o***x value*] [**–q**[*time*]]
[**–r***name*] [**–t**] [**–v**] [address **...**]

DESCRIPTION

Sendmail sends a message to one or more people, routing the message over whatever networks are necessary. *Sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to a control-D or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, for example, if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

OPTIONS

–ba Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.

–bd Run as a daemon, waiting for incoming SMTP connections.

–bi Initialize the alias database.

–bm Deliver mail in the usual way (default).

–bp Print a summary of the mail queue.

–bs Use the SMTP protocol as described in RFC821. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.

–bt Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

–bv Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

–C*file* Use alternate configuration file.

–d*X* Set debugging value to *X*.

–F*fullname* Set the full name of the sender.

–f*name* Sets the name of the "from" person (that is, the sender of the mail). **–f** can only be used by "trusted" users (who are listed in the config file).

–h*N* Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop.

–M*id* Attempt to deliver the queued message with message-id **id**.

–n Don't do aliasing.

–o*xvalue* Set option *x* to the specified *value*. Options are described below.

–q[*time*] Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *Time* is given as a tagged number, with 's' being seconds, 'm' being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, "—q1h30m" or "—q90m" would both set the timeout to one hour thirty minutes.

—r*name* An alternate and obsolete form of the —f flag.

—R*string* Go through the queue of pending mail and attempt to deliver any message with a recipient containing the specified string. This is useful for clearing out mail directed to a machine which has been down for awhile.

—t Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for people to send to. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed.

—v Go into verbose mode. Alias expansions will be announced, etc.

PROCESSING OPTIONS

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **—o** flag or in the configuration file. These are described in detail in the *Installation and Operation Guide*. The options are:

A*file* Use alternate alias file.

c On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.

d*x* Set the delivery mode to *x*. Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only — that is, actual delivery is done the next time the queue is run.

D Try to automatically rebuild the alias database if necessary.

e*x* Set error processing to mode *x*. Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.

F*mode* The mode to use when creating temporary files.

f Save UNIX-style From lines at the front of messages.

g*N* The default group id to use when calling mailers.

H*file* The SMTP help file.

i Do not take dots on a line by themselves as a message terminator.

L*n* The log level.

m Send to "me" (the sender) also if I am in an alias expansion.

o If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (that is, commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.

Q*queuedir* Select the directory in which to queue messages.

r*timeout* The timeout on reads; if none is set, *sendmail* will wait forever for a mailer.

S*file* Save statistics in the named file.

s — Always instantiate the queue file, even under circumstances where it is not strictly necessary.

T*time* — Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, they will be returned to the sender. The default is three days.

t*stz,dtz* — Set the name of the time zone.

u*N* — Set the default user id for mailers.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep *sendmail* from suppressing the blanks from between arguments.

Sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases, sendmail* rebuilds the alias database. If invoked as *mailq, sendmail* prints the contents of the mail queue.

FILES

Except for */usr/lib/sendmail.cf,* these pathnames are all specified in */usr/lib/sendmail.cf.* Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	data base of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/bin/uux	to deliver uucp mail
/bin/mail	to deliver local mail
/usr/spool/mqueue/*	temp files and queued mail
~/.mailcf	Individual users' configuration file

SEE ALSO

biff(1), binmail(1), mail(1), aliases(5),

DARPA Internet Request For Comments RFC819, RFC821, RFC822,

In the *System Manager's Manual*:

Setting Up the Mail System in the *System Set-up and Operation* chapter.
Sendmail – An Internetwork Mail Router, in the Tutorials section
Sendmail Installation and Operation Guide, in the Tutorials section

BUGS

Sendmail converts blanks in addresses to dots. This is incorrect according to the old ARPANET mail protocol RFC733 (NIC 41952), but is consistent with the new protocols (RFC822).

NAME

sendnews – send news articles via mail

SYNOPSIS

sendnews [**−o**] [**−a**] [**−b**] [**−n** newsgroups] destination

DESCRIPTION

sendnews reads an article from it's standard input, performs a set of changes to it, and gives it to the mail program to mail it to *destination*.

An 'N' is prepended to each line for decoding by *uurec*(1).

OPTIONS

−o handle old format articles.

−a used for sending articles via the ARPANET. It maps the article's path from *uucphost!xxx* to *xxx@arpahost*.

−b used for sending articles via the **Berknet**. It maps the article's path from *uucphost!xxx* to *berkhost:xxx*.

−n change the article's newsgroup to the specified *newsgroup*.

SEE ALSO

inews(1), uurec(8), recnews(8), readnews(1), checknews(1)

NAME

showmount − show all remote mounts

SYNOPSIS

showmount [**−a**] [**−d**] [**−e**] [host]

DESCRIPTION

Showmount lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the *mountd*(8c) server on *host*, and is saved across crashes in the file */etc/rmtab*. The default value for *host* is the value returned by *hostname*(1).

OPTIONS

−d List directories that have been remotely mounted by clients.

−a Print all remote mounts in the format

```
hostname:directory
```

where *hostname* is the name of the client, and *directory* is the root of the file system that has been mounted.

−e Print the list of exported file systems.

SEE ALSO

rmtab(5), mountd(8), exports(5)

BUGS

If a client crashes, its entry will not be removed from the list until it reboots and executes *umount −a*.

NAME

shutdown – close down the system at a given time

SYNOPSIS

/etc/shutdown [**–k**] [**–r**] [**–h**] time [warning-message ...]

DESCRIPTION

Shutdown provides an automated shutdown procedure for the super-user to notify users politely when the system is shutting down. *Time* specifies when *shutdown* will bring the system down; it may be the word **now** (indicating an immediate shutdown), or it may specify a future time in one of two formats: *+number* and *hour:min.* The first form brings the system down in *number* minutes, and the second brings the system down at the time of day indicated in 24-hour notation.

At intervals that get closer as the apocalypse approaches, warning messages are displayed at terminals of all logged-in users, and of users who have remote mounts on that machine. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating */etc/nologin* and writing a message there. If this file exists when a user attempts to log in, *login*(1) prints its contents and exits. The file is removed just before *shutdown* exits.

At shutdown time a message is written in the file */usr/adm/shutdownlog*, containing the time of shutdown, the instigator of the shutdown, and the reason. Then a terminate signal is sent to *init*, which brings the system down to single-user mode.

The time of the shutdown and the warning message are placed in */etc/nologin*, which should be used to inform the users as to when the system will be back up, and why it is going down (or anything else).

OPTIONS

As an alternative to the above procedure, these options can be specified:

–r Execute *reboot*(8).

–h Execute *halt*(8).

–k Make people think the system is going down, but do not actually take it down.

FILES

/etc/nologin tells login not to let anyone log in
/usr/adm/shutdownlog log file for succesful shutdowns.

SEE ALSO

login(1), reboot(8)

BUGS

Only allows you to kill the system between now and 23:59 if you use the absolute time for shutdown. @(#)skyversion.8 1.3 85/04/05 SMI

NAME

skyversion – print the SKYFFP board microcode version number

SYNOPSIS

/usr/etc/skyversion

DESCRIPTION

skyversion obtains from the SKYFFP board the Sky version number of the microcode currently loaded and prints the result on the standard output.

DIAGNOSTICS

The Sky version number operation code used to implement this command is not available for microcode releases earlier than Sky release 3.00. The result in this case is unpredictable and is either a nonmeaningful version number or a message indicating that no version number is available. Meaningful version numbers are of the form *n.dd* where $n \geq 3$.

NAME

sticky – executable files with persistent text

DESCRIPTION

While the 'sticky bit', mode **01000** (see *chmod*(2)), is set on a sharable executable file, the text of that file will not be removed from the system swap area. Thus the file does not have to be fetched from the file system upon each execution. As long as a copy remains in the swap area, the original text cannot be overwritten in the file system, nor can the file be deleted. Directory entries can be removed so long as one link remains.

Sharable files are made by the **–z** option of *ld*(1).

To replace a sticky file that has been used do: (1) Clear the sticky bit with *chmod*(1). (2) Execute the old program to flush the swapped copy. This can be done safely even if others are using it. (3) Overwrite the sticky file. If the file is being executed by any process, writing will be prevented; it suffices to simply remove the file and then rewrite it, being careful to reset the owner and mode with *chmod* and *chown*(2). (4) Set the sticky bit again.

Only the super-user can set the sticky bit.

NAME

swapon — specify additional device for paging and swapping

SYNOPSIS

/usr/etc/swapon −a
/usr/etc/swapon name ...

DESCRIPTION

Swapon is used to specify additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to *swapon* normally occur in the system multi-user initialization file */etc/rc* making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

Normally, the **−a** argument is given, causing all devices marked as "sw" swap devices in **/etc/fstab** to be made available.

The second form gives individual block devices as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

SEE ALSO

swapon(2), init(8)

FILES

/dev/[ru][pk]?b normal paging devices

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to insure all disk writes have been completed before the processor is halted in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8)

NAME

sysdiag – system diagnostics

SYNOPSIS

/usr/diag/sysdiag/sysdiag

DESCRIPTION

Sysdiag is a general-purpose system diagnostic facility that tests the system and reports its findings. It concentrates on three areas of system functionality; memory, peripherals and disk.

To use *sysdiag,* log on as *sysdiag,* then enter the command *sysdiag.*

Sysdiag creates a Suntools environment with one window each for memory, peripherals, and disk error messages, plus a window for the console. It also creates date/time and performance monitor graphs. It places abbreviated error messages from the memory, disk, and peripherals in the appropriate windows, and sends console messages to the console window.

When called from a terminal *sysdiag* interleaves all its messages on the screen.

With or without the windows, it places long error messages in files named **log.***xx.nn* where:

xx	is the name of diagnostic
nn	is the pass number (increments each pass)

After it completes its test, *sysdiag* displays the error log files by executing the command **more log****. These files remain after *sysdiag* exits.

Sysdiag consists of a user account with a home directory, a collection of scripts, and executable files containing the actual test code.

To configure or change *sysdiag,* either change the shell commands in */usr/diag/sysdiag/sysdiag,* or change the *sysdiag* user configuration files *.login, .suntools,* and *.cshrc.*

SEE ALSO

Sun-2/160 Diagnostics Manual and *Sun-2/120 Diagnostics Manual.*

NAME

syslog – log systems messages

SYNOPSIS

/usr/etc/in.syslog [**–m***N*] [**–f***name*] [**–d**] [**–p** *port*]

DESCRIPTION

Syslog reads a datagram socket and logs each message it reads into a set of files described by the configuration file */etc/syslog.conf*. *Syslog* configures when it starts up and whenever it receives a hangup signal. *Syslog* logs to the host specified by 'loghost' in the */etc/hosts* file. For details on running *syslog* in a Sun network environment, see the section, "System Log Configuration" in the *System Set-up and Operation* chapter of the *System Installation and Maintenance Guide.*

Each message logged consists of one line. A message can contain a priority code, marked by a digit in angle braces at the beginning of the line. Priorities are defined in <syslog.h>, as defined in the list below. *LOG_ALERT* is prioity 1 (the highest priority) while *LOG_DEBUG* is priority 9 (the lowest priority).

LOG_ALERT — this priority should essentially never be used. It applies only to messages that are so important that every user should be aware of them, for example, a serious hardware failure.

LOG_SALERT — messages of this priority should be issued only when immediate attention is needed by a qualified system person, for example, when some valuable system resource disappears. They get sent to a list of system people.

LOG_EMERG — Emergency messages are not sent to users, but represent major conditions. An example might be hard disk failures. These could be logged in a separate file so that critical conditions could be easily scanned.

LOG_ERR — these represent error conditions, such as soft disk failures, etc.

LOG_CRIT — such messages contain critical information, but which can not be classed as errors, for example, 'su' attempts. Messages of this priority and higher are typically logged on the system console.

LOG_WARNING — issued when an abnormal condition has been detected, but recovery can take place.

LOG_NOTICE — something that falls in the class of "important information;" this class is informational but important enough that you don't want to throw items in it away casually. Messages without any priority assigned to them are typically mapped into this priority.

LOG_INFO — information level messages. These messages could be thrown away without problems, but should be included if you want to keep a close watch on your system.

LOG_DEBUG — it may be useful to log certain debugging information. Normally this will be thrown away.

It is expected that the kernel will not log anything below LOG_ERR priority.

The *syslog* configuration file, *etc/syslog.conf,* consists of two sections separated by a blank line. The first section defines files that *syslog* will log into. Each line contains a single digit which defines the lowest priority (highest numbered priority) that this file will receive, an optional asterisk which guarantees that something gets output at least every 20 minutes, and a pathname. The second part of the file contains a list of users that will be informed on SALERT level messages. For example, the configuration file:

```
5*/dev/tty8
8/usr/spool/adm/syslog
```

```
3/usr/adm/critical

eric
kridle
kalash
```

logs all messages of priority 5 or higher onto the system console, including timing marks every 20 minutes; all messages of priority 8 or higher into the file */usr/spool/adm/syslog*; and all messages of priority 3 or higher into */usr/adm/critical.* The users 'eric', 'kridle', and 'kalash' will be informed on any subalert messages.

OPTIONS

−m *N* Set the mark interval to *N* (default 20 minutes).

−f *name*
Specify an alternate configuration file.

−d Turn on debugging (if compiled in).

−p *port*
Port number where *syslog* listens for incoming datagrams. The default port is defined in the 'syslog/udp' entry in the */etc/services* file.

To bring *syslog* down, it should be sent a terminate signal. It logs that it is going down and then waits approximately 10 seconds for any additional messages to come in.

There are some special messages that cause control functions. '<*>N' sets the default message priority to *N.* '<$>' causes *syslog* to reconfigure (equivalent to a hangup signal). This can be used in a shell file run automatically early in the morning to truncate the log.

Syslog creates the file /etc/syslog.pid if possible containing a single line with its process id. This can be used to kill or reconfigure *syslog.*

FILES

/etc/hosts – the hosts file
/etc/syslog.conf – the configuration file
/etc/syslog.pid – the process id
/etc/services – to find the *syslog* server's port number.

BUGS

LOG_ALERT and LOG_SALERT messages should only be allowed to privileged programs.

Actually, *syslog* is not clever enough to deal with kernel error messages in the current implementation.

SEE ALSO

syslog(3), kill(2)

NAME

talkd – server for talk program

SYNOPSIS

/usr/etc/in.talkd

DESCRIPTION

Talkd is a server used by the *talk(1)* program. It listens at the udp port indicated in the "talk" service description; see *services*(5). The actual conversation takes place on a tcp connection that is established by negotiation between the two machines involved.

SEE ALSO

services(5), talk(1), inetd(8)

BUGS

The protocol is architecture dependent, and can't be relied upon to work between Suns and other machines.

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/usr/etc/in.telnetd host.port

DESCRIPTION

Telnetd is a server which supports the DARPA standard TELNET virtual terminal protocol. The TELNET server is invoked by *inetd*(8C) each time there is a connection to the telnet service; see *services*(5).

Telnetd operates by allocating a pseudo-terminal device (see *pty*(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. *Telnetd* manipulates the master side of the pseudo terminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, *telnetd* sends a TELNET option to the client side indicating a willingness to do "remote echo" of characters. The pseudo terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*(4)). Aside from this initial setup, the only mode changes *telnetd* will carry out are those required for echoing characters at the client side of the connection.

Telnetd supports binary mode, and most of the common TELNET options, but does not, for instance, support timing marks.

SEE ALSO

telnet(1C)

BUGS

A complete list of the options supported should be given here.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

/usr/etc/in.tftpd [**–d**] [*port*]

DESCRIPTION

Tftpd is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(5), and is invoked each time a datagram reaches this port by the internet server *inetd*(8C).

Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. To do this, **tftpd** executes as *uid –2*, *gid –2* , assuming that no files exist with that owner or group. However, nothing check this assumption or enforces this restriction.

SEE ALSO

tftp(1C)

BUGS

This server is known only to be self consistent (i.e. it operates with the user TFTP program, *tftp*(1C)).

NAME

timed – DARPA Time server

SYNOPSIS

/usr/etc/in.timed

DESCRIPTION

Timed is a server which supports the DARPA Time Server Protocol. The time server operates at the port indicated in the "time" service description; see *services*(5), and is invoked by *inetd*(8C) each time there is a connection to the time server.

SEE ALSO

services(5), rdate(8), inetd(8)

BUGS

A more sophisticated facility that can accept broadcasts and synchronize clocks over an internet is needed.

NAME

trpt – transliterate protocol trace

SYNOPSIS

/usr/etc/trpt [**−a**] [**−s**] [**−t**] [**−j**] [**−p***hex-address*] [system [core]]

DESCRIPTION

Trpt interrogates the buffer of TCP trace records created when a socket is marked for 'debugging' (see *setsockopt*(2)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB).

OPTIONS

−s Print a detailed description of the packet sequencing information, in addition to the normal output.

−t Print the values for all timers at each point in the trace, in addition to the normal output.

−j Just give a list of the protocol control block addresses for which there are trace records.

−p*hex-address*
Show only trace records associated with the protocol control block who's address follows.

−a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **−A** option to *netstat*(8). Then run *trpt* with the **−p** option, supplying the associated protocol control block addresses. If there are many sockets using the debugging option, the **−j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/vmunix
/dev/kmem

SEE ALSO

setsockopt(2), netstat(8)

DIAGNOSTICS

'no namelist' when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the trace record.

The output format is inscrutable, and should be described here.

NAME

tunefs – tune up an existing file system

SYNOPSIS

/etc/tunefs [**–a** *maxcontig*] [**–d** *rotdelay*] [**–e** *maxbpg*] [**–m** *minfree*] *special* | *filesys*

DESCRIPTION

Tunefs is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the flags given below:

–a *maxcontig*

This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see –d below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

–d *rotdelay*

This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

–e *maxbpg*

This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

–m *minfree*

This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

SEE ALSO

fs(5), newfs(8), mkfs(8), dumpfs(8)

The Sun UNIX File System in the Sun *System Internals Manual.*

BUGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems. (if run on the root file system, the system must be rebooted)

NAME

update — periodically update the super block

SYNOPSIS

/etc/update

DESCRIPTION

Update is a program that executes the *sync*(2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO

sync(2), sync(8), init(8)

NAME

uuclean — uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [**−p***pre*] [**−n***time*] [**−m**]

DESCRIPTION

Uuclean scans the spool directory for files with the specified prefix and deletes all those which are older than the specified number of hours.

OPTIONS

−p*pre* Scan for files with *pre* as the file prefix. Up to 10 **−p** arguments may be specified. A **−p** without any *pre* following deletes all files older than the specified time.

−n*time* Files whose age is more than *time* hours are deleted if the prefix test is satisfied (default time is 72 hours).

−m Send mail to the owner of the file when it is deleted.

Uuclean will typically be started by *cron*(8).

FILES

/usr/lib/uucp	directory with commands used by uuclean internally
/usr/lib/uucp/spool	spool directory

SEE ALSO

uucp(1C), uux(1C)

NAME

uurec – receive processed news articles via mail

SYNOPSIS

uurec

DESCRIPTION

uurec reads news articles on the standard input sent by *sendnews*(8), decodes them, and gives them to *inews*(1) for insertion.

SEE ALSO

inews(1), readnews(1), recnews(8), sendnews(8), newscheck(1)

NAME

vipw – edit the password file

SYNOPSIS

/etc/vipw

DESCRIPTION

Vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The *vi* editor will be used unless the environment variable EDITOR indicates an alternate editor. *Vipw* performs a number of consistency checks on the password entry for *root*, and will not allow a password file with a "mangled" root entry to be installed.

SEE ALSO

chsh(1), passwd(1), passwd(5), adduser(8)

FILES

/etc/ptmp

NAME

vmstat – report virtual memory statistics

SYNOPSIS

vmstat [**–fsS**] [*interval* [*count*]]

DESCRIPTION

Vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and cpu activity.

Without options, *vmstat* displays a one-line summary of the virtual memory activity since the system has been booted. If *interval* is specified, *vmstat* summarizes activity over the last *interval* seconds. If a *count* is given, the statistics are repeated *count* times.

For example, the following command displays a summary of what the system is doing every five seconds. This is a good choice of printing interval since this is how often some of the statistics are sampled in the system.

```
hal% vmstat 5

procs   memory                    page     disk      faults     cpu
r b w  avm  fre  re at  pi  po  fr  de  sr x0 x1 x2 x3  in  sy  cs us  sy id
2 0 0  918  286  0  0   0   0   0   0   0  1  0  0  0   4  12   5  3   5 91
1 0 0  846  254  0  0   0   0   0   0   0  6  0  1  0  42 153  31  7  40 54
1 0 0  840  268  0  0   0   0   0   0   0  5  0  0  0  27 103  25  8  26 66
1 0 0  620  312  0  0   0   0   0   0   0  6  0  0  0  26  76  25  6  27 67

^C
hal%
```

The fields of *vmstat*'s display are:

procs Reports the number of processes in each of the three following states:
r in run queue
b blocked for resources (i/o, paging, etc.)
w runnable or short sleeper (< 20 secs) but swapped

memory Reports on usage of virtual and real memory. Virtual memory is considered active if it belongs to processes which are running or have run in the last 20 seconds.
avm number of active virtual Kbytes
fre size of the free list in Kbytes

page Reports information about page faults and paging activity. The information on each of the following activities is averaged each five seconds, and given in units per second.
re page reclaims — but see the **–S** option for how this field is modified.
at number of attaches — but see the **–S** option for how this field is modified.
pi pages paged in
po pages paged out
fr pages freed per second
de anticipated short term memory shortfall
sr pages scanned by clock algorithm, per-second

disk Reports number of disk operations per second (this field is system dependent). For Sun systems, four slots are available for up to four drives: "x0" (or "s0" for SCSI disks), "x1", "x2", and "x3".

faults Reports trap/interrupt rate averages per second over last 5 seconds.
in (non clock) device interrupts per second
sy system calls per second
cs cpu context switch rate (switches/sec)

cpu Gives a breakdown of percentage usage of CPU time.

us	user time for normal and low priority processes
sy	system time
id	cpu idle

OPTIONS

−f Report on the number of *forks* and *vforks* since system startup and the number of pages of virtual memory involved in each kind of fork.

−s Display the contents of the *sum* structure, giving the total number of several kinds of paging-related events which have occurred since boot.

−S Report on swapping rather than paging activity. This option will change two fields in *vmstat*'s "paging" display: rather than the "re" and "at" fields, *vmstat* will report "si" (swap-ins), and "so" (swap-outs).

FILES

/dev/kmem
/vmunix

NAME

ypinit - build and install yellow pages database

SYNOPSIS

ypinit −m
ypinit −s master_name

DESCRIPTION

Ypinit sets up a yellow pages (YP) server's database. It can be used to set up a master server or a slave server. You must be the superuser to run it. It asks a few questions, which are self-explanatory, and reports success or failure to the terminal.

It sets up a master server using the simple model in which that server is master to all maps in the data base. This is the way to bootstrap the YP system; later if you want you can change the association of maps to masters by changing the information in the YP map named *ypmaps*. All databases are built from scratch, either from information available to the program at runtime, or from the ASCII data base files in */etc*. The list of which files will be used to generate the databases is below in the FILES section. All of these files should be in their traditional form, rather than the abbreviated form used on client machines, which primarily use the YP services to access the same information.

A slave YP server database is set up by copying an existing database from a running server. The *master_name* argument should be the host name of YP server, which is either actually the master server for all the maps, or a server the data base of which is believed to be up-to-date and stable. If your network uses different server machines acting as the master of different maps, of course no single machine will be the master of all maps. This doesn't matter; the yp data bases will converge automatically in time, and the process can be sped up somewhat by using *yppush*(8) and *yppull*(8). Just choose a YP server which is up, reachable, and stable.

OPTIONS

−m Indicates that the local host is to be the YP master.

−s Set up a slave database.

FILES

/etc/passwd
/etc/group
/etc/hosts
/etc/networks
/etc/services
/etc/protocols
/etc/netgroup

SEE ALSO

makedbm(8), ypfiles(8), yppush(8), ypmake(8)

NAME

ypmake – rebuild yellow pages database

SYNOPSIS

cd /etc/yp ; make

DESCRIPTION

There is a *Makefile* in */etc/yp* for building the yellow pages database. With no arguments, *make* creates *dbm* databases for everything that is out of date, and then executes *yppush*(8) to notify slave databases that there has been a change.

Typing *make passwd* will create and *yppush* just the password database (assuming it is out of date). Likewise, *make hosts* and *make networks* will create and *yppush* the host and network files, */etc/hosts* and */etc/networks*.

There are three special variables: DIR, which gives the directory of the source files; NOPUSH, which when non-null inhibits doing a *yppush* of the new database files; and DOM, used to construct a domain other than the master's default domain. The default for DIR is */etc*, and the default for NOPUSH is the null string.

SEE ALSO

makedbm(8), ypserv(8)

NAME

yppasswdd – server for modifying yellow pages password file

SYNOPSIS

/usr/etc/rpc.yppasswdd file [**–m** arg1 arg2 ...]

DESCRIPTION

Yppasswdd is a server that handles password change requests from *yppasswd*(1). It changes a password entry in *file*, which is assumed to be in the format of *passwd*(5). An entry in *file* will only be changed if the password presented by *yppasswd*(1) matches the encrypted password of that entry.

If the **–m** option is given, then after *file* is modified, a *make*(1) will be performed in */etc/yp*. Any arguments following the flag will be passed to *make*.

This server is not run by default, nor can it be started up from *inetd*(8). If it is desired to enable remote password updating for the yellow pages, then an entry for *yppasswdd* should be put in the */etc/rc* file of the host serving as the master for the yellow pages *passwd* file.

EXAMPLE

If the yellow pages password file is stored as */etc/yp/src/passwd*, then to have password changes propagated immediately, the server should be invoked as

```
/usr/etc/rpc.yppasswdd /etc/yp/src/passwd –m passwd DIR=/etc/yp/src
```

FILES

/etc/yp/Makefile

SEE ALSO

yppasswd(1), passwd(5), ypfiles(5), ypmake(8)

CAVEAT

This server will eventually be replaced with a more general service for modifying any map in the yellow pages

NAME

yppush, yppull, yppoll - yellow pages administration utilities

SYNOPSIS

yppush [**−h** hostname] [**−d** domainname] mapname
yppull [**−h** hostname] [**−d** domainname] mapname
yppoll [**−h** hostname] [**−d** domainname] mapname

DESCRIPTION

These commands supply an interface to the running yellow pages (YP) system. They make use of YP services themselves, which must be available at the site where the commands are issued. The issuing site may be a yp client; it need not be a server. The *rpcinfo*(8) program may be useful for determining if the YP processes *ypserv* and *ypbind* are up and running at a site. The processes that comprise the YP system are described in *ypserv*(8). The YP database format is described in *ypfiles*(5).

Yppush tells a master YP server process (*ypserv*) to direct its peer processes to set the master of the named map to that master server, and to get a new copy of the named map. If the host is not the master of the named map, the command will succeed, but no action will be taken by the YP server.

Yppull tells a slave YP server process (*ypserv*) to try and get a more recent copy of the named map, preferably from the map's master. If the server is the master of the named map, the command will succeed, but no action will be taken by the YP server.

Yppoll asks a *ypserv* (master or slave) for information about one map. The information returned is whether the map exists at the site, what version number is in the map, and which server is thought to be the map's master.

OPTIONS

−h Specifies the *hostname* on which the *ypserv* process is running. The default is the local host.

−d Specifies an alternate *domainname* for the YP. The default is the domain for the local host.

SEE ALSO

rpcinfo(8), ypserv(8), ypfiles(5)

NAME

ypserv – yellow pages server and binder processes

SYNOPSIS

/etc/ypserv
/etc/ypbind

DESCRIPTION

The yellow pages (YP) provides a simple network lookup service. YP consists of databases and processes. The databases are *dbm*(3) files in a directory tree rooted at */etc/yp*, described in *ypfiles*(5). The processes are */etc/ypserv*, the YP database server, and */etc/ypbind*, the YP binder. The programmatic interface to YP is described in *ypclnt*(3N). Administrative tools are described in *yppush*(8) and *ypcat*(1). Database generation and maintenance tools are described in *ypinit*(8), *ypmake*(8), and *makedbm*(8).

Both *ypserv* and *ypbind* are daemons: they are typically activated at system startup time from */etc/rc.local*, and theoretically run forever. *Ypserv* runs only on YP server machines with a complete YP database. *Ypbind* runs on all machines using YP services, both YP client machines and YP servers.

The *ypserv* daemon has two functions: to look up information in its local database (or YP maps), and to keep that database consistent with those of its peers.

The operations performed by *ypserv* are defined in the header file *<rpcsvc/yp_prot.h>*. Communication to and from *ypserv* is by means of RPC calls. Lookup functions are supplied (as a C library) in the *ypclnt* package, defined in the header file *<rpcsvc/ypclnt.h>* and described in *ypclnt*(3N). There are three types of functions: *match*, *get first*, and *get next*. The *match* operation takes a key, and returns the associated value within a named map and domain. (Domains are described in *ypfiles (8).)* The *get first* operation returns the first key-value pair within a named map and domain, and *get next* can be used to enumerate the remainder. There are other operations performed by *ypserv:* they either give back information internal to the process itself, or are useful in speeding up the update propagation algorithm which is discussed in the next paragraph. Non-lookup functions are accessible to programmers by coding the RPC calls defined in *<rpcsvc/yp_prot.h>*, and at the shell level by using the tools *yppush*, *yppull*, and *yppoll*, all described in *yppush*(8).

This paragraph discusses how map updates propagate among YP servers, and how propagation converges. Each map is associated with a version (or order) number. Each *ypserv* process communicates with its peers, and tries to find the map availible with the greatest order number. The order number is set at the time the map is is generated. Each map is associated with a distinguished YP server host, called the map's master. Updates to that map should be done only at the map's master — maps should be generated only on their master host. As each *ypserv* tries to find the version of any map containg the greatest order number, it first tries to communicate with the map's master. If the master is unreachable, it chooses another peer at random, and will transfer that map from any peer that has a map containing an order number greater than the one in the map it currently has.

This paragraph discusses the *ypbind* process, whose function is to remember information that lets clients on a single node communicate with *ypserv* processes. *Ypbind* must be running on every network node; *ypserv* may or may not be running on the same node, but must be running somewhere on the network. The information *ypbind* remembers is called a binding — the association of a domain name with the internet address of the YP server, and the port on that host at which the *ypserv* process is listening for service requests. The process of binding is driven by client requests. As a request for an unbound domain comes in, the *ypbind* process broadcasts on the net trying to find a *ypserv* process that claims to serve maps within that domain. Since binding is accomplished by broadcasting, there must be at least one *ypserv* process on every net. Once a domain is bound by a particular *ypbind*, that same binding is given to every client process on the node running that *ypbind*. The binding is verified before it is given out to a client process. If *ypbind* is unable to speak to the *ypserv* process using its binding it marks the domain as unbound, tells the client process that the domain is unbound, and tries again to bind the domain. Requests for binding received by an unbound domain will fail immediately.

SEE ALSO

ypclnt(3N), ypfiles(5), makedbm(8), ypcat(8), ypinit(8), ypmake(8), yppush(8)

NAME

ypwhich – which machines are the YP server and master?

SYNOPSIS

ypwhich [**–d** domainname] [hostname]

ypwhich [**–d** domainname] [**–t**] **–m** [mname]

DESCRIPTION

Ypwhich tells which YP server supplies yellow pages to a YP client, and which YP server is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP master it is using.

If the **–m** switch is used without *mname,* a list of every map in the domain and the master of each will be printed. If *mname* is specified, only the master YP server for that map is printed. *Mname* may be a mapname, or a nickname for a mapname. Mapnames and nicknames are described in *ypcat*(8).

OPTIONS

–d *Domainname* specifies the name of a YP domain. The default is the default domain for the local machine.

–m Find the master YP server for a map, or for all maps in a domain. No *hostname* may be specified with **–m.**

–t Inhibit nickname translation; useful if there is a mapname identical to a nickname. This is not true of any Sun-supplied map.

SEE ALSO

ypfiles(5), rpcinfo(8), yppush(8), ypserv(8)

Commands Reference Manual

Index

Special Characters
%job & — job to background — csh, 72
%job — job to foreground — csh, 72
. shell command — sh, 292
: shell command — sh, 292
@ — arithmetic on shell variables — csh, 72

A
abort printer — lpc, 526
ac — login accounting, 478
login accounting — ac, 478
accounting on or off, 583
accton — accounting on or off, 583
adb — debugger, 2
adb scripts — adbgen, 479
adbgen — generate adb script, 479
addbib — create bibliography, 9
address resolution display and control — arp, 484
adduser — add new user account, 481
adjacentscreens, 11
admin — adminster SCCS, 12
administration
  install — install files, 176
adventure — exploration game, 409
alias — shell macros — csh, 66
alias substitution
  in C-Shell, 61
alloc — show memory use — csh, 66
allow messages — mesg, 212
alter process priority — renice, 563
analyze — crash analyzer, 483
ar — library maintenance, 16
archive
  ar — library maintenance, 16
archive tapes — tar, 333
archives
  copy, 55
  cpio — copy archives, 55
argument list processing
  in C-Shell, 75 *thru* 76
arithmetic — drill in number facts, 410
arp — address resolution display and control, 484
as — assembler, 18
ASCII dump file — od, 224
ascii — ASCII character set, 458
at — do job at specified time, 19
autoboot procedures — reboot, 561
awk — scan and process patterns, 20

B
backgammon — backgammon game, 411
backspace magnetic tape files — mt, 218
backspace magnetic tape records — mt, 218
backup dumps — dump, 503
banner — large banner, 413
basename — strip filename affixes, 23
bballs — black and white demo, 415
bbounce — black and white demo, 415
bc — calculator language, 24
bcd — convert to antique media, 414
bdemos — black and white demo, 415
bdraw — interactive graphics drawing, 426
bg — job to background — csh, 66
bibliographic references

- bibliographic references, *continued*
  - **refer** — insert literature references, 267
- bibliography management
  - create or extend, 9
  - **indxbib** — make inverted index, 173
  - **lookbib** — find bibliographic references, 188
  - **sortbib** — sort bibliographic database, 305
- **biff** — mail notifier, 26
- binary file transmission
  - **uudecode** — decode binary file, 370
  - **uuencode** — encode binary file, 370
- **binmail** — version 7 mail, 27
- **biod** — NFS daemon, 549
- **bjump** — black and white demo, 415
- black and white demos
  - **bbounce**, 415
  - **bdemos**, 415
  - **bjump**, 415
  - **bphoto**, 415
  - **brotcube**, 415
- blocks
  - count, in file — **sum**, 315
- **boggle** — boggle game, 416
- bootstrap procedures — **reboot**, 561
- bootstrap PROM monitor program, 536
- Bourne Shell — **sh**, 292
  - **:**, 296
  - **break**, 296
  - **case**, 292
  - **cd**, 296
  - **continue**, 296
  - **eval**, 296
  - **exec**, 296
  - **exit**, 296
  - **export**, 296
  - **false** command, 132
  - **for**, 292
  - **if**, 292
  - **login**, 296
  - **newgrp**, 296
  - **read**, 296
  - **readonly**, 296
  - **set**, 296
  - **shift**, 297
  - **times**, 297
  - **trap**, 297
  - **umask**, 297
  - **wait**, 297
  - **while**, 293
- **bphoto** — black and white demo, 415
- **break** shell command — **sh**, 296
- **break** — exit loop — **csh**, 66
- **breaksw** — exit switch — **csh**, 66
- broadcast messages
  - **rwall** — to all users on network, 277, 391
- **brotcube** — black and white demo, 415
- **bsuncube** — display 3-D Sun logo, 417
- make random library — **ranlib**, 258
- build system configuration files — **config**, 489
- build yellow pages database — **ypinit**, 611

C

- C compiler, 33
- C language
  - **indent** — format C source, 170
- C programming language
  - **ctags** — create tags file, 78
  - **mkstr** — create C error messages, 214
  - **tcov** — code coverage tool, 338
  - **vgrind** — make formatted C listings, 377
  - **xstr** — extract strings from C code, 403
- C programming languages
  - **lint** — C program verifier, 183, 183
- C Shell variables, 72 *thru* 75
  - **argv**, 72
  - **cdpath**, 73
  - **cwd**, 73
  - **echo**, 73
  - **histchars**, 73
  - **history**, 73
  - **home**, 73
  - **ignoreeof**, 73
  - **mail**, 73
  - **noclobber**, 73
  - **noglob**, 73
  - **nonomatch**, 73
  - **notify**, 73
  - **path**, 73
  - **prompt**, 74
  - **savehist**, 74
  - **shell**, 74
  - **status**, 74
  - **time**, 74
  - **verbose**, 75
- C-Shell commands
  - **%job** — job to foreground, 72
  - **%job &** — job to background, 72
  - **@** — arithmetic on shell variables, 72
  - **alias** — shell macros, 66
  - **alloc** — show memory use, 66
  - **bg** — job to background, 66
  - **break** — exit loop, 66
  - **breaksw** — exit switch, 66
  - **case** — selector in switch, 66
  - **cd** — change directory, 66
  - **chdir** — change directory, 66
  - **continue** — cycle loop, 67
  - **default** — catchall in switch, 67
  - **echo** — echo arguments, 67
  - **else** — alternative commands, 67
  - **end** — end loop, 67
  - **endif** — end conditional, 67
  - **endsw** — end switch, 67

C-Shell commands, *continued*
    eval — re-evaluate shell data, 67
    exec — execute command, 67
    exit — exit shell, 67
    fg — job to foreground, 67
    foreach — loop on list of names, 67
    glob — filename expand argument list, 67
    goto — command transfer, 68
    hashstat — display hashing statistics, 68
    history — display history list, 68
    if — conditional statement, 68
    jobs — display job list, 68
    kill — kill jobs and processes, 68
    limit — alter resource limitations, 69
    login — login new user, 69
    logout — end session, 69
    nice — run low priority process, 69
    nohup — run command immune to hangups, 69
    notify — request immediate notification, 69
    onintr — handle interrupts in scripts, 69
    popd — pop shell directory stack, 70
    pushd — push shell directory stack, 70
    rehash — recompute command hash table, 70
    repeat — execute command repeatedly, 70
    set — change value of shell variable, 70
    setenv — set variable in environment, 70
    shift — shift argument list, 70
    source — read commands from file, 71
    stop — halt job or process, 71
    suspend — suspend shell, 71
    switch — multi-way branch, 71
    time — time command, 71
    umask — change/display file creation mask, 71
    unalias — remove aliases, 71
    unhash — discard hash table, 71
    unlimit — remove resource limitiations, 71
    unset — discard shell variables, 72
    unsetenv — remove environment variables, 72
    wait — wait for background process, 72
    while — repeat commands, 72
cal — display calendar, 29
calculator, 95
calendar — reminder service, 30
call graph
    display profile data, 153
canfield — canfield solitaire card game, 418
case shell command — sh, 292
case — selector in switch — csh, 66
cat — concatenate files, 31
catman — create cat files for manual, 485
cb — C beautifier, 32
cballs — color demo, 423
cc — C compiler, 33
ccat — compress files, 53
cd — change directory, 36
cd shell command — sh, 296
cd — change directory — csh, 66
cdc — change delta commentary, 37
cdraw — color demo, 423, 426
change
    default shell, 43
    delta commentary, 37
    directory, 36
    group, 41
    login shell, 43
    mode of file, 42
    permissions of file, 42
    working directory, 36
change login password — passwd, 227
change name of directory — mv, 219
change name of file — mv, 219
change priority of command — nice, 220
change process priority — renice, 563
character translation — tr, 352
characters for equations — eqnchar, 459
characters in file
    count — wc, 392
chase — escape killer robots, 419
chdir — change directory — csh, 66
check directory — dcheck, 497
check file system — fsck, 508
check spelling — spell, 306
checkeq — check *eqn* constructs, 119
checknews — check for news, 39
checknr — check nroff/troff files, 40
chess — chess game, 420
chesstool — chess game, 421
chgrp — change group, 41
ching — book of changes, 422
chmod — change mode, 42
chown — change owner of file, 486
chsh — change shell, 43
clean print queue — lpc, 526
clean UUCP spool area — uuclean, 606
clear — clear screen, 44
clear i-node — clri, 487
clocktool — display time in window, 46
clri — clear i-node, 487
cmp — compare files, 47
code coverage tool — tcov, 338
code formatter
    indent — format C source, 170
col — filter reverse paper motions, 48
colcrt — document previwer, 49
color demo
    cballs, 423
    cdraw, 423

- color demo, *continued*
  - cphoto, 423
  - cpipes, 423
  - cshowmap, 423
  - csnow, 423
  - csuncube, 423
  - csunlogo, 423
  - cvlsi, 423
- colrm — remove columns from file, 50
- columns
  - print in multiple — pr, 239
  - remove from file, 50
- comb — combine deltas, 51
- combine SCCS deltas, 51
- comm — find commonality, 52
- command
  - change priority of — nice, 220
  - describe — whatis, 394
  - locate — whereis, 395
  - run immune to hangup — nohup, 220
- command execution
  - in C-Shell, 75
- command interpreter
  - sh — Bourne Shell, 292
- command substitution
  - in C-Shell, 63
- commands
  - introduction, 1
- communications
  - cu — connect to remote system, 346
  - enroll — enroll for secret mail, 402
  - mail — send and receive mail, 200
  - permit or deny messages — mesg, 212
  - talk — talk to another user, 332
  - telnet — TELNET interface, 342
  - tip — connect to remote system, 346
  - uuclean — clean UUCP spool area, 606
  - uucp — UNIX to UNIX copy, 368
  - uudecode — decode binary file, 370
  - uuencode — encode binary file, 370
  - uulog — UUCP log, 368
  - uusend — send file to remote host, 371
  - uux — UNIX to UNIX command execution, 372
  - write — write to another user, 400
  - xget — receive secret mail, 402
  - xsend — send secret mail, 402
- compare
  - files, 47
  - files differentially, 105
  - three-way differential, 107
- compare versions of SCCS file — sccsdiff, 284
- compile Pascal programs — pc, 228
- compiler generator, 128, 404
- compiler generators
  - lex — generate lexical analyzer, 182
- compilers
  - cc — C compiler, 33
- compress — compress files, 53
- compress files, 53
- comsat — biff server, 488
- concatenate files
  - cat — concatenate files, 31
- config — build system configuration files, 489
- configuration files
  - build — config, 489
- configure network interface parameters — ifconfig, 519
- connect to remote system — cu, 346
- connect to remote system — tip, 346
- continue shell command — sh, 296
- continue — cycle loop — csh, 67
- control line printer — lpc, 526 *thru* 527
- control magnetic tape — mt, 218
- convert
  - spaces to tabs, 125
  - tabs to spaces, 125
- convert and copy files, 97
- convert foreign font files — vswap, 386
- convert units — units, 365
- copy
  - archives, 55
  - files, 54
- copy files from remote machine — rcp, 261
- copy standard output to many files — tee, 339
- core images
  - get of process, 147
- count blocks in file — sum, 315
- count lines, words, characters in file — wc, 392
- cp — copy files, 54
- cphoto — color demo, 423
- cpio — copy archives, 55
- cpipes — color demo, 423
- CPU PROM monitor program, 536
- crash analyzer — analyze, 483
- crash — crash procedure, 494
- create
  - bibliography, 9
  - delta, 99
  - SCCS data bases, 12
  - SCCS delta, 99
  - tags file, 78
- create cat files for manual — catman, 485
- create directory — mkdir, 213
- create error log — dmesg, 502
- create file system — mkfs, 533
- create font width table — vwidth, 388
- create mail aliases database — newaliases, 547
- create new file system — newfs, 548
- create permuted index — ptx, 252

create prototype file system — **mkproto**, 535
make random library — **ranlib**, 258
create script of terminal session — **script**, 288
create special file — **mknod**, 534
create system configuration files — **config**, 489
create system log entry — **syslog**, 330
create yellow pages database — **ypinit**, 611
create yellow pages dbm file — **makedbm**, 530
cribbage — cribbage card game, 424
cron — clock daemon, 496
cross reference Pascal program — **pxref**, 257
crypt — encrypt, 57
csh — C shell, 58 *thru* 77
cshowmap — color demo, 423
csnow — color demo, 423
csuncube — color demo, 423
csunlogo — color demo, 423
ctags — create tags file, 78
cu — connect to remote system, 346
current domain
  set or display, 108
curve fitting
  spline — interpolate smooth curve, 307
cvlsi — color demo, 423

D

DARPA Internet directory service — **whois**, 399
DARPA TELNET protocol server — **telnetd**, 600
DARPA Time server — **timed**, 602
DARPA to RPC mapper — **portmap**, 553
Data Encryption Standard, 102
database operator — **join**, 177
date — date and time, 79
dbx — debugger, 80
dbxtool — debugger, 90
dc — desk calculator, 95
dcheck — directory consistency check, 497
dd — convert and copy, 97
debug network — **ping**, 552
debug tools
  adb — debugger, 2
  adbgen — generate adb script, 479
  dbx — debugger, 80
  dbxtool — debugger, 90
decimal dump file — **od**, 224
decode binary file — **uudecode**, 370
decode files, 57
crypt — decrypt, 57, 102
decrypt files, 57
default — catchall in switch — **csh**, 67
delete
  columns from file, 50
  filename affixes, 23
delete, *continued*
  nroff, troff, tbl and eqn constructs, 101
delete directory — **rmdir**, 273
delete outdated news articles — **expire**, 506
delete print jobs — **lprm**, 194
delete repeated lines — **uniq**, 364
delta
  change commentary, 37
  combine, 51
  remove — **rmdel**, 272
delta — make delta, 99
demount file system — **umount**, 539
deny messages — **mesg**, 212
deroff — remove troff constructs, 101
des — data encryption, 102
describe command — **whatis**, 394
desk calculator, 95
df — display free space, 104
diag — initialize/diagnose disk, 498
diagnose/initialize disk — **diag**, 498
diagnostics
  gxtest — graphics board diagnostics, 515
  imemtest — memory diagnostic, 520
diagnostics — **sysdiag**, 596
diff — differential compare, 105
diff3 — three-way differential compare, 107
directory
  change name of — **mv**, 219
  change working, 36
  check consistency — **dcheck**, 497
  delete — **rm**, 271
  delete — **rmdir**, 273
  differential compare, 105
  display name of working — **pwd**, 253
  erase — **rm**, 271
  erase — **rmdir**, 273
  list contents of — **ls**, 196
  make — **mkdir**, 213, 214
  move — **mv**, 219
  purge — **rm**, 271
  purge — **rmdir**, 273
  remove — **rm**, 271
  remove — **rmdir**, 273
  rename — **mv**, 219
disable print queue — **lpc**, 526
disk
  dkinfo — geometry information, 501
disk diagnose/initialize — **diag**, 498
disk diagnostics — **sysdiag**, 596
display
  call graph profile data, 153
  current domain, 108
  current host identifier, 165
  current host name, 166
  date, 79
  date and time, 79
  disk usage, 109

display, *continued*
  first lines of file, 163
  free space in file system, 104
  group membership, 162
  identifier of current host, 165
  name of current host, 166
  time and date, 79
  time in window, 46
display editor — vi, 379
display effective user name — whoami, 398
display file names — ls, 196
display last commands — lastcomm, 180
display last part of file — tail, 331
display name list — nm, 221
display online manuals — man, 211
display page size — pagesize, 226
display printer queue — lpq, 190
display process status — ps, 247
display program profile — prof, 242
display SCCS file editing status — sact, 279
display status of local hosts — ruptime, 276
display system up time — uptime, 366
display users on system — users, 367
display waiting mail — prmail, 241
display working directory name — pwd, 253
dkinfo — disk geometry information, 501
dmesg — create error log, 502
document preparation
  bibliography management, 9
document production
  checknr — check nroff/troff files, 40
  col — filter reverse paper motions, 48
  colcrt command, 49
  delete nroff, troff, tbl and eqn constructs, 101
  eqnchar — special characters for equations, 459
  equation processor, 119
  indxbib — make inverted index, 173
  lookbib — find bibliographic references, 188
  man — macros to format manual pages, 467
  mathematical formatting, 119
  me — macro package, 469
  ms — macro package, 471
  nroff — document formatter, 222
  ptx — generate permuted index, 252
  refer — insert literature references, 267
  remove nroff, troff, tbl and eqn constructs, 101
  set equations, 119
  simple formatter, 136
  soelim — eliminate .so's from nroff input, 302
  sortbib — sort bibliographic database, 305
  spell — check spelling, 306
document production, *continued*
  tbl — table formatter, 337
  troff — typeset documents, 353
  vfontinfo — examine font files, 376
  vtroff — format document for raster printer, 387
  vwidth — make font width table, 388
domain
  set or display current, 108
domainname — set/display domain name, 108
draw graph, 157
du — display disk usage, 109
dump — dump file system, 503
dump frame buffer image — screendump, 285
dumpfs — dump file system information, 505

E

echo — echo arguments, 110
echo — echo arguments — csh, 67
ed — line editor, 111
edit
  fonts, 138
edit — line editor, 123
edit password file — vipw, 608
editing text
  edit — line editor, 123
  ex — line editor, 123
  sed — stream editor, 289
editing text files
  ed — line editor, 111
editor
  icon, 167
egrep — pattern scanner, 159
eliminate .so's from nroff input — soelim, 302
else — alternative commands — csh, 67
emulate Tektronix 4014 — tektool, 340
enable print queue — lpc, 526
encode binary file — uuencode, 370
encode files, 57
encrypt, 102
encrypt files, 57
encrypted mail
  enroll for — enroll, 402
  receive — enroll, 402
  send — xsend, 402
encryption key
  generate — makekey, 532
end — end loop — csh, 67
endif — end conditional — csh, 67
endsw — end switch — csh, 67
enroll — enroll for secret mail, 402
environment
  display variables — printenv, 240
  tset — set terminal characteristics for, 356
environment variables
  in C-Shell, 72 *thru* 75

eqn
  remove constructs, 101
eqn — mathematical typesetting, 119
eqnchar — special characters for equations, 459
erase directory — rmdir, 273
erase file or directory — rm, 271
erase magnetic tape — mt, 218
error — analyze error messages, 121
eval shell command — sh, 296
eval — re-evaluate shell data — csh, 67
evaluate expressions, 126
ex — line editor, 123
exec shell command — sh, 296
exec — execute command — csh, 67
execute commands at specified times — cron, 496
executing non-builtin commands
  in C-Shell, 75
exit shell command — sh, 296
exit — exit shell — csh, 67
expand — expand tabs, 125
expire — remove outdated news articles, 506
export shell command — sh, 296
expr — evaluate expressions, 126
expression evaluation, 126
extend
  bibliography, 9
extract strings from C code — xstr, 403
eyacc — compiler generator, 128

F

f77 — FORTRAN compiler, 129
fastboot — reboot system, 507
fasthalt — halt system, 507
fg — job to foreground — csh, 67
fgrep — pattern scanner, 159
file
  browse through text— more, 215
  browse through text— page, 215
  change name of — mv, 219
  change ownership — chown, 486
  copy from remote machine — rcp, 261
  count lines, words, characters in — wc, 392
  delete — rm, 271
  display last part of — tail, 331
  dump — od, 224
  erase — rm, 271
  find lines in sorted — look, 187
  identify version — what, 393
  move — mv, 219
  print — lpr, 192
  purge — rm, 271
  remove — rm, 271
  rename — mv, 219
  reverse lines in — rev, 268
  send to remote host — uusend, 371
file, *continued*
  split into pieces — split, 308
  sum — sum and count blocks in file, 315
  update last modified date of — touch, 351
file — get file type, 133
file system
  cd — change directory, 36
  check and repair — fsck, 508
  check consistency — icheck, 518
  check directory — dcheck, 497
  create new — newfs, 548
  demount — umount, 539
  display free space, 104
  dump information — dumpfs, 505
  free space display, 104
  make — mkfs, 533
  make prototype— mkproto, 535
  mount — mount, 539
  summarize ownership — quot, 558
  tune — tunefs, 604
  unmount — umount, 539
  where am I — pwd, 253
file system dump — dump, 503
file system hierarchy — hier, 460
file system restore — restore, 564
file transfer protocol
  ftp command, 144
file transfer protocol server — ftpd, 511
filename substitution
  in C-Shell, 63 *thru* 65
files
  basename — strip affixes, 23
  cat — concatenate, 31
  ccat — compress files, 53
  chmod — change mode, 42
  cmp — compare files, 47
  colrm — remove columns from, 50
  comm — find commonality, 52
  compare, 47
  compare three-way differential, 107
  compress — compress files, 53
  convert and copy, 97
  copy, 54
  copy standard output to many — tee, 339
  cp — copy files, 54
  cpio — copy archives, 55
  crypt — encrypt/decrypt, 57
  decode, 57
  decrypt, 57
  determine type of, 133
  differential compare, 105
  display first lines of, 163
  display names — ls, 196
  encode, 57
  encrypt, 57
  find, 134
  find differences, 105
  prepare files for printing — pr, 239

files, *continued*
  search for patterns in, 159
  sort — sort or merge files, 303
  split FORTRAN file, 143
  three-way differential compare, 107
  transfer, 144
  uncompress — uncompress files, 53
filter
  reverse paper motions, 48
find — find files, 134
find lines in sorted file — **look**, 187
find literature references — **refer**, 267
find object file size — **size**, 300
find ordering for object library — **lorder**, 189
find patterns in file, 159
find printable strings in binary file — **strings**, 309
find program — **whereis**, 395
fish — Go Fish game, 428
flush disk activity — **sync**, 329
fmt — simple formatter, 136
fold — fold long lines, 137
font
  vwidth — make font width table, 388
font files
  convert foreign — **vswap**, 386
fonttool — font editor, 138
for shell command — **sh**, 292
foreach — loop on list of names — **csh**, 67
format document for raster printer — **vtroff**, 387
format Lisp programs — **vlp**, 382
format tables — **tbl**, 337
FORTRAN
  f77 — FORTRAN compiler, 129
  print file, 141
  split file, 143
FORTRAN language
  ratfor — rational FORTRAN, 260
FORTRAN programming language
  ctags — create tags file, 78
fortune — get fortune, 429
fpr — print FORTRAN file, 141
from — who is mail from, 142
fsck — check and repair file system, 508
fsirand — install random inode generation numbers, 510
fsplit — split FORTRAN file, 143
ftp — file transfer, 144
ftpd — file transfer protocol server, 511

G

gammontool — backgammon game, 430
gcore — core image of process, 147
generate adb script — **adbgen**, 479
generate encryption key — **makekey**, 532
generate lexical analyzer — **lex**, 182
generate Pascal execution profile — **pxp**, 255
generate permuted index — **ptx**, 252
get — get SCCS file, 148
get magnetic tape unit status — **mt**, 218, 218
get terminal name — **tty**, 361
get terminal options — **stty**, 311
gettable — get NIC host tables, 513
getty — set terminal mode, 514
gfxtool, 154
glob — filename expand argument list — **csh**, 67
goto — command transfer — **csh**, 68
gprof — call graph profile, 153
graph — draw graph, 157
graphics
  spline — interpolate smooth curve, 307
  vplot — plot on Versatec, 383
graphics board diagnostics — **gxtest**, 515
graphics filters — **plot**, 236
graphics tool, 154
grep — pattern scanner, 159
group
  change, 41
groups — display group membership, 162
gxtest — graphics board diagnostics, 515

H

halt — stop processor, 516
halt system — **fasthalt**, 507
hangman — hangman game, 432
hashstat — display hashing statistics — **csh**, 68
head — display head of file, 163
help — get SCCS help, 164
hexadecimal dump file — **od**, 224
hier — file system hierarchy, 460
history — display history list — **csh**, 68
hostid — display host ID, 165
hostname — display host name, 166
htable — convert NIC standard format host tables, 517

I

i-node
  clear — **clri**, 487
I/O statistics report — **iostat**, 524
icheck — file system consistency check, 518
icontool, 167
identify file version — **what**, 393
if shell command — **sh**, 292
if — conditional statement — **csh**, 68
ifconfig — configure network interface parameters, 519
imemtest — memory diagnostic, 520

incremental file system dump — dump, 503
incremental file system restore — restore, 564
indent — format C source, 170
indexing
    ptx — generate permuted index, 252
indxbib — make inverted index, 173
inetd — internet services daemon, 521
inews — submit news articles, 174
information handling
    awk — scan and process patterns, 20
    bc — calculator language, 24
inhibit messages — mesg, 212
init — process control initialization, 522
initialize/diagnose disk — diag, 498
insert literature references — refer, 267
install — install files, 176
install yellow pages database — ypinit, 611
interactive graphics drawing — bdraw, 426
Internet directory service — whois, 399
Internet file transfer protocol server — ftpd, 511
interpolate smooth curve — spline, 307
interpret Pascal — px, 254
introduction to commands, 1
introduction to miscellaneous information, 457
iostat — report I/O statistics, 524

J

jobs — display job list — csh, 68
join — relational database operator, 177

K

kgmon — dump profile buffers, 525
kill — terminate process, 178
kill — kill jobs and processes — csh, 68

L

language
    vgrind — make formatted C listings, 377
languages
    cc — C compiler, 33
    f77 — FORTRAN compiler, 129
    indent — format C source, 170
    lex — generate lexical analyzer, 182
    lint — C program verifier, 183
    mkstr — create C error messages, 214
    pc — Pascal compiler, 228, 233, 235
    pmerge — merge Pascal files, 237
    px — interpret Pascal, 254
    pxp — generate Pascal execution profile, 255
    pxref — cross reference Pascal program, 257
    ratfor — rational FORTRAN, 260
    tcov — code coverage tool, 338
    vlp — format Lisp programs, 382
languages, *continued*
    xstr — extract strings from C code, 403
last — list last logins, 179
lastcomm — display last commands, 180
leave — remind you of leaving time, 181
lex — generate lexical analyzer, 182
library
    find ordering for object — lorder, 189
library management
    ar — library maintenance, 16
    ranlib — make random library, 258
limit — alter resource limitations — csh, 69
line printer control — lpc, 526 *thru* 527
line printer daemon — lpd, 528
lines
    find, in sorted file — look, 187
lines in file
    count — wc, 392
lint — C program verifier, 183
Lisp programming language
    vlp — format Lisp programs, 382
literature references
    find and insert — refer, 267
load frame buffer image — screenload, 286
locate program — whereis, 395
lockscreen — save window context, 185
login
    change password — passwd, 227
    display effective user name — whoami, 398
    list last — last, 179
    make script of session — script, 288
    rwho — who is on local network, 278
    save window context — lockscreen, 185
    to remote machine — rlogin, 269
    what are users doing — w, 389
    who — who is logged in, 397
login accounting — ac, 478
login — sign on, 186, 296
login environment
    display variables — printenv, 240
    tset — set terminal characteristics, 356
    tty — get terminal name, 361
login — login new user — csh, 69
logout — end session — csh, 69
look — find lines in a sorted file, 187
look for pattern in file, 159
lookbib — find bibliographic references, 188
lorder — find ordering for object library, 189
lpc — line printer control, 526
lpd — line printer daemon, 528
lpq — display printer queue, 190
lpr — print files, 192
lprm — remove print jobs, 194
ls — list files, 196

M

m4 — macro processor, 198
macro processor — m4, 198
magnetic tape
  backspace files — mt, 218
  backspace records — mt, 218
  erase — mt, 218
  forward space files — mt, 218
  forward space records — mt, 218
  get unit status — mt, 218
  manipulate — mt, 218
  place unit off-line — mt, 218
  retension — mt, 218
  rewind — mt, 218
  skip backward files — mt, 218
  skip backward records — mt, 218
  skip forward files — mt, 218
  skip forward records — mt, 218
  write EOF mark on — mt, 218
magnify raster image — rastrepl, 259
mail
  comsat — biff server, 488
  enroll for secret — enroll, 402
  print waiting — prmail, 241
  receive secret mail — enroll, 402
  send secret mail — xsend, 402
mail aliases
  create database — newaliases, 547
mail — send and receive mail, 200
mail delivery — sendmail, 586
mail services
  biff — mail notifier, 26
  binmail — version 7 mail, 27
  who is mail from, 142
mailaddr — mail addressing, 464
maintenance and operation, 475
make
  delta, 99
  SCCS delta, 99
make — build programs, 208
make directory — mkdir, 213
make file system — mkfs, 533
make font width table — vwidth, 388
make formatted C listings — vgrind, 377
make mail aliases database — newaliases, 547
make new file system — newfs, 548
make permuted index — ptx, 252
make prototype file system — mkproto, 535
make random library — ranlib, 258
make script of terminal session — script, 288
make special file — mknod, 534
make system configuration files — config, 489
make system log entry — syslog, 330
make system special files — makedev, 531
make yellow pages database — ypinit, 611
make yellow pages dbm file — makedbm, 530
makedbm — make yellow pages dbm file, 530
makedev — make system special files, 531
makekey — generate encryption key, 532
man — display online manuals, 211
man — macros to format manual pages, 467
manipulate magnetic tape — mt, 218
manipulate routing tables — route, 573
manual pages — catman — create cat files for, 485
  describe command — whatis, 394
manuals
  display online — man, 211
me — macro package, 469
memory diagnostic — imemtest, 520
memory diagnostics — sysdiag, 596
merge or sort files — sort, 303
merge Pascal files — pmerge, 237
mesg — permit or deny messages, 212
messages
  permit or deny — mesg, 212
mille — Mille Bornes game, 433
miscellaneous information, 457
mkdir — make directory, 213
mkfs — make file system, 533
mknod — make special file, 534
mkproto — make prototype file system, 535
mkstr — create C error messages, 214
modes
  chmod — change mode, 42
monitor program, 536
monop — Monopoly game, 436
more — browse text file, 215
mount — mount file system, 539
mount file system — mount, 539
mountd — NFS mount server, 541
move directory — mv, 219
move file — mv, 219
move print jobs — lpc, 526
ms — macro package, 471
mt — manipulate magnetic tape, 218
multiple columns
  print in — pr, 239
mv — move or rename files or directory, 219

N

ncheck — convert i-numbers to filenames, 542
nd — network disk control, 543
neqn — mathematical typesetting, 119
netstat — display network status, 545
network
  copy files across — rcp, 261
  rwall — write to all users, 277
  rwho — who is logged in on local network, 278

network debugging — **ping**, 552
network disk control — **nd**, 543
network interface parameters
  configure — **ifconfig**, 519
network news
  checknews — check for news, 39
  inews — submit news articles, 174
  read articles — **readnews**, 262
  receive unprocessed articles — **recnews**, 266
  submit articles — **postnews**, 238
network routing daemon — **routed**, 574
network rwall server — **rwalld**, 580
network status
  display — **netstat**, 545
newaliases — make mail aliases database, 547
newfs — make new file system, 548
newgrp shell command — **sh**, 296
news
  checknews — check for news, 39
  read articles — **readnews**, 262
  receive unprocessed articles — **recnews**, 266
  submit articles — **postnews**, 238
news articles
  receive unprocessed, 562
  remove outdated — **expire**, 506
news delivery — **sendnews**, 589
NFS mount server — **mountd**, 541
NFS statistics
  display — **nfsstat**, 550
nfsd — NFS daemon, 549
nfsstat — display network statistics, 550
NIC host tables
  get from host — **gettable**, 513
nice — change priority of command, 220
nice — run low priority process — **csh**, 69
nm — display name list, 221
nohup — run command immune to hangup, 220
nohup — run command immune to hangups — **csh**, 69
non-builtin command execution
  in C-Shell, 75
notify — request immediate notification — **csh**, 69
nroff
  checknr — check nroff/troff files, 40
  col — filter reverse paper motions, 48
  colcrt command, 49
  remove constructs, 101
nroff — document formatter, 222
number — convert Arabic numerals to English, 438

O

object code management
  ar — library maintenance, 16
  ranlib — make random library, 258
object file
  find printable strings in — **strings**, 309
  size — find object file size, 300
  strip — strip symbols and relocation bits, 310
object library
  find ordering for — **lorder**, 189
octal dump file — **od**, 224
od — dump file, 224
onintr — handle interrupts in scripts — **csh**, 69
online manuals
  display — **man**, 211
owner of file
  chown — change, 486

P

pac — printer/plotter accounting, 551
page — browse text file, 215
page size
  display — **pagesize**, 226
pagesize — display page size, 226
parser generator — **yacc**, 404
Pascal compiler — **pc**, 228
Pascal interpreter code translator — **pc**, 233
Pascal language
  pmerge — merge Pascal files, 237
  px — interpret Pascal, 254
  pxp — generate Pascal execution profile, 255
  pxref — cross reference Pascal program, 257
Pascal programming language
  ctags — create tags file, 78
Pascal translator and interpreter — **pc**, 235
passwd — change login password, 227
password
  change login — **passwd**, 227
password file
  edit — **vipw**, 608
patterns
  search in file for, 159
pc — Pascal compiler, 228
perfmeter — display performance statistics, 231
perfmon — display performance statistics, 232
performance monitoring — **perfmeter**, 231, 232
  prof — display program profile, 242
  pxp — generate Pascal execution profile, 255
  time — time command, 344

performance tools
    display call graph profile data, 153
peripheral diagnostics — sysdiag, 596
permissions
    chmod — change mode, 42
permit messages — mesg, 212
permuted index
    generate — ptx, 252
pi — Pascal interpreter code translator, 233
ping — debug network, 552
pix — Pascal translator and interpreter, 235
place magnetic tape unit off-line — mt, 218, 218
plot — graphics filters, 236
plot on Versatec — vplot, 383
pmerge — merge Pascal files, 237
popd — pop shell directory stack — csh, 70
portmap — DARPA to RPC mapper, 553
postmortem crash analyzer — analyze, 483
postnews — submit news articles, 238
pr — prepare files for printing, 239
predefined variables
    in C-Shell, 72 *thru* 75
prepare files for printing — pr, 239
pretty print
    indent — format C source, 170
pretty printer
    vgrind — make formatted C listings, 377
    vlp — format Lisp programs, 382
preview documents
    colcrt command, 49
print
    FORTRAN file, 141
print files
    lpr — print files, 192
print values from YP database — ypcat, 406
print waiting mail — prmail, 241
print working directory name — pwd, 253
printenv — display environnment, 240
printer
    abort — lpc, 526
    clean queue — lpc, 526
    disable queue — lpc, 526
    lpq — display queue, 190
    enable queue — lpc, 526
    move jobs — lpc, 527
    remove jobs from queue — lprm, 194
    restart — lpc, 526
    start — lpc, 526
    status of — lpc, 526
    stop — lpc, 526
printer control — lpc, 526 *thru* 527
printer daemon — lpd, 528
printer/plotter accounting, 551
prmail — print waiting mail, 241
process
    change priority — renice, 563
process, *continued*
    display status — ps, 247
    terminate — kill, 178
    wait — wait process completion, 390
processes
    get core image of, 147
prof — display program profile, 242
profile
    display call graph, 153
profiling
    prof — display program profile, 242
    pxp — generate Pascal execution profile, 255
programming languages
    analyze and disperse compiler error messages, 121
    assembler, 18
    cc — C compiler, 33
    f77 — FORTRAN compiler, 129
    lex — generate lexical analyzer, 182
    lint — C program verifier, 183
    pc — Pascal compiler, 228, 233, 235
    pmerge — merge Pascal files, 237
    print FORTRAN file, 141
    px — interpret Pascal, 254
    pxp — generate Pascal execution profile, 255
    pxref — cross reference Pascal program, 257
    split FORTRAN file, 143
    vgrind — make formatted C listings, 377
    vlp — format Lisp programs, 382
    xstr — extract strings from C code, 403
programming tools
    adb — debug tool, 2
    bc — calculator language, 24
    cb — C beautifier, 32
    compiler generator, 128
    ctags — create tags file, 78
    dbx — debugger, 80
    dbxtool — debugger, 90
    display call graph profile data, 153
    indent — format C source, 170
    install — install files, 176
    lex — generate lexical analyzer, 182
    lint — C program verifier, 183
    lorder — find ordering for object library, 189
    m4 — macro processor, 198
    maintain object libraries, 16
    make — build programs, 208
    mkstr — create C error messages, 214
    nm — display name list, 221
    prof — display program profile, 242
    ranlib — make random library, 258
    ratfor — rational FORTRAN, 260
    sccs — source code control, 280
    size — find object file size, 300

- programming tools, *continued*
  - **strings** — find printable strings in binary file, 309
  - **strip** — strip symbols and relocation bits, 310
  - **tcov** — code coverage tool, 338
  - **time** — time command, 344
  - **touch** — update last modified date of file, 351
  - **yacc** — compiler generator, 404
- programs
  - introduction, 1
- PROM monitor program, 536
- provide truth values — **true**, 355
- **prs** — display SCCS history, 243
- **ps** — display process status, 247
- **pstat** — display system statistics, 554
- **ptx** — generate permuted index, 252
- purge directory — **rmdir**, 273
- purge file or directory — **rm**, 271
- **pushd** — push shell directory stack — **csh**, 70
- **pwd** — print working directory name, 253
- **px** — interpret Pascal, 254
- **pxp** — generate Pascal execution profile, 255
- **pxref** — cross reference Pascal program, 257

Q

- queue
  - **lpq** — display printer, 190
  - remove jobs from printer — **lprm**, 194
- **quiz** — test knowledge, 439
- **quot** — summarize file system ownership, 558

R

- **rain** — display raindrops, 440
- **ranlib** — make random library, 258
- **rastrepl** — magnify raster image, 259
- **ratfor** — rational FORTRAN, 260
- **rc** — startup commands, 559
- **rcp** — remote file copy, 261
- **rdate** — remote date, 560
- **read** shell command — **sh**, 296
- read mail — **mail**, 200
- read news articles — **readnews**, 262
- **readnews** — read news articles, 262
- **readonly** shell command — **sh**, 296
- **reboot** — system startup procedures, 561
- reboot system — **fastboot**, 507
- rebuild yellow pages database — **ypmake**, 612
- receive secret mail — **enroll**, 402
- receive unprocessed articles — **recnews**, 266
- receive unprocessed news articles — **recnews**, 562
- **recnews** — receive unprocessed articles, 266, 562
- **refer** — insert literature references, 267
- **rehash** — recompute command hash table — **csh**, 70
- relational database operator — **join**, 177
- reminder services
  - **biff** — mail notifier, 26
  - **calendar** — reminder service, 30
  - **leave** — remind you of leaving time, 181
- remote execution server — **rexecd**, 567
- remote file copy — **rcp**, 261
- remote host
  - send file to — **uusend**, 371
- remote login — **rlogin**, 269
- remote login server — **rlogind**, 569
- remote magtape protocol server — **rmt**, 571
- remote shell — **rsh**, 274
- remote shell server — **rshd**, 577
- remote system cu
  - connect to — **cu**, 346
- remote system tip
  - connect to — **tip**, 346
- remove
  - columns from file, 50
  - delta — **rmdel**, 272
  - filename affixes, 23
  - nroff, troff, tbl and eqn constructs, 101
- remove delta from SCCS file — **rmdel**, 272
- remove directory — **rmdir**, 273
- remove outdated news articles — **expire**, 506
- remove print jobs — **lprm**, 194
- remove repeated lines — **uniq**, 364
- rename directory — **mv**, 219
- rename file — **mv**, 219
- **renice** — change process priority, 563
- **repeat** — execute command repeatedly — **csh**, 70
- **reset** — reset terminal bits, 356
- reset terminal bits — **reset**, 356
- restart printer — **lpc**, 526
- **restore** — restore file system, 564
- restore file system — **restore**, 564
- restore frame buffer image — **screenload**, 286
- retension magnetic tape — **mt**, 218
- **rev** — reverse lines in file, 268
- reverse lines in file — **rev**, 268
- rewind magnetic tape — **mt**, 218
- **rexecd** — remote execution server, 567
- **rlogin** — remote login, 269
- **rlogind** — remote login server, 569
- **rmail** — process remote mail, 570
- **rmdel** — remove delta from SCCS file, 272
- **rmdir** — remove directory, 273
- **rmt** — remote magtape protocol server, 571
- **route** — manipulate routing tables, 573
- **routed** — network routing daemon, 574

routing tables
    manipulate — **route**, 573
rpcinfo — report RPC information, 576
rsh — remote shell, 274
rshd — remote shell server, 577
rstatd — kernel statistics server, 579
ruptime — display status of local hosts, 276
rwalld — network rwall server, 580
rwho — who is logged in on local network, 278
rwhod — system status server, 581

S

sa — accounting summary, 583
sail — wooden ships and iron men, 441
savecore — save OS core dump, 585
sccs — source code control system, 280
SCCS commands
    admin — administer SCCS, 12
    cdc — change delta commentary, 37
    comb — combine deltas, 51
    get — get SCCS file, 148
    help — get SCCS help, 164
    cdc — display SCCS history, 243
    rmdel — remove delta, 272
    sact — display SCCS file editing status, 279
    sccsdiff — compare versions of SCCS file, 284
    unget — unget SCCS file, 363
    val — validate SCCS file, 373
SCCS delta
    change commentary, 37
    combine, 51
    create, 99
    remove — **rmdel**, 272
SCCS history
    display, 243
sccsdiff — compare versions of SCCS file, 284
screen fonts
    edit, 138
screen-oriented editor — **vi**, 379
screendump — dump frame buffer image, 285
screenload — restore frame buffer image, 286
script — make script of terminal session, 288
search for files, 134
search for pattern in file, 159
secret mail
    enroll for — **enroll**, 402
    receive — **enroll**, 402
    send — **xsend**, 402
sed — stream editor, 289
send and receive mail — **mail**, 200
send file to remote host — **uusend**, 371
send secret mail — **xsend**, 402
sendmail — mail delivery system, 586
sendnews — news delivery, 589
servers
    biod — NFS daemon, 549
    comsat — biff server, 488
    ftpd — Internet File Transfer Protocol, 511
    inetd — internet services daemon, 521
    mountd — mount request server, 541
    nfsd — NFS daemon, 549
    rexecd — remote execution server, 567
    rlogind — remote login server, 569
    rshd — remote shell server, 577
    rstatd — kernel statistics server, 579
    rwalld — network rwall server, 580
    rwhod — system status server, 581
    talkd — talk program server, 599
    telnetd — TELNET protocol server, 600
    tftpd — Trivial File Transfer Protocol server, 601
    timed — DARPA Time server, 602
    yppasswdd — yellow pages password server, 613
set
    current domain, 108
    current host name, 166
    name of current host, 166
set shell command — **sh**, 296
set — change value of shell variable — **csh**, 70
set terminal characteristics — **tset**, 356
set terminal options — **stty**, 311
setenv — set variable in environment — **csh**, 70
sh — Bourne Shell, 292
Shell
    Bourne — **sh**, 292
shell
    change login, 43
    chsh — change shell, 43
    csh — C shell, 58 *thru* 77
    remote — **rsh**, 274
Shell scripts
    sun — test for Sun Workstation, 316
    true — provide truth values, 355
    vax — test machine is VAX, 375
shells
    false command, 132
shelltool, 297
shift shell command — **sh**, 297
shift — shift argument list — **csh**, 70
showmount — display remote mounts, 590
shutdown — shut down system, 591
login — sign on, 186
    to remote machine — **rlogin**, 269
signal handling
    in C-Shell, 76
signon
    signon last — **last**, 179
size — find object file size, 300
skip backward magnetic tape files — **mt**, 218

skip backward magnetic tape records — **mt**, 218
skip forward magnetic tape files — **mt**, 218
skip forward magnetic tape records — **mt**, 218
skyversion — display SKY version, 592
sleep — suspend execution, 301
smoothing
    spline — interpolate smooth curve, 307
snake — display chase game, 451
soelim — eliminate .so's from nroff input, 302
sort bibliographic database — **sortbib**, 305
sort — sort or merge files, 303
sort or merge files — **sort**, 303
sort topologically — **tsort**, 360
sortbib — sort bibliographic database, 305
sorted file
    find lines in — **look**, 187
    remove repeated lines — **uniq**, 364
source code control system — **sccs**, 280
source — read commands from file — **csh**, 71
spaces
    unexpand to tabs, 125
special characters for equations — **eqnchar**, 459
special file
    make — **mknod**, 534
special files — **makedev**, 531
spell — check spelling, 306
spellin — check spelling, 306
spellout — check spelling, 306
spline — interpolate smooth curve, 307
split — split file into pieces, 308
stand-alone utilities
    diag — initialize/diagnose disk, 498
    gxtest — graphics board diagnostics, 515
    imemtest — memory diagnostic, 520
standard output
    copy to many files — **tee**, 339
start printer — **lpc**, 526
startup procedures — **reboot**, 561
statistics
    I/O — **iostat**, 524
    rstatd — kernel statistics server, 579
statistics of NFS
    display — **nfsstat**, 550
status of network
    display — **netstat**, 545
status of printer — **lpc**, 526
sticky — set sticky bit, 593
stop printer — **lpc**, 526
stop processor — **halt**, 516
stop — halt job or process — **csh**, 71
stream editor — **sed**, 289
strings — find printable strings in binary file, 309
strip
strip, *continued*
    filename affixes, 23
strip — strip symbols and relocation bits, 310
stty — get terminal options, 311
stty — set terminal options, 311
su — substitute user id, 314
submit news articles, 174, 238
substitute user id — **su**, 314
sum — sum and count blocks in file, 315
sun — test for Sun Workstation, 316
Sun Workstation
    test for — **sun**, 316
suntools, 316
suntools — Suntools window environment, 318
SunWindows
    graphics tool, 154
    icon tool, 167
    shell tool, 297
    start up environment, 316
suspend execution — **sleep**, 301
suspend — suspend shell — **csh**, 71
swapon — specify paging device, 594
switch — multi-way branch — **csh**, 71
sync — update super block, 329, 595
sysdiag — system diagnostics, 596
syslog — make system log entry, 330, 597
system administration
    adduser — add new user account, 481
    analyze — crash analyzer, 483
    catman — create cat files for manual, 485
    install — install files, 176
system configuration files
    build — **config**, 489
system maintenance and operation, 475
system PROM monitor program, 536
system special files — **makedev**, 531
system status server — **rwhod**, 581

T

tabs
    expand to spaces, 125
tail — display last part of file, 331
talk — talk to another user, 332
talkd — talk server, 599
tape
    backspace files — **mt**, 218
    backspace records — **mt**, 218
    erase — **mt**, 218
    forward space files — **mt**, 218
    forward space records — **mt**, 218
    get unit status — **mt**, 218
    manipulate magnetic — **mt**, 218
    place unit off-line — **mt**, 218
    retension — **mt**, 218
    rewind — **mt**, 218
    skip backward files — **mt**, 218

- tape, *continued*
  - skip backward records — mt, 218
  - skip forward files — mt, 218
  - skip forward records — mt, 218
  - write EOF mark on — mt, 218
- tape archiver — tar, 333
- tar — tape archiver, 333
- tbl
  - remove constructs, 101
- tbl — table formatter, 337
- tcov — code coverage tool, 338
- tee — copy standard output to many files, 339
- tektool — emulate Tektronix 4014, 340
- Tektronix 4014
  - emulate — tektool, 340
- telnet — TELNET interface, 342
- telnetd — TELNET protocol server, 600
- terminal
  - get name of — tty, 361
  - get options — stty, 311
  - make script of session — script, 288
  - reset bits — reset, 356
  - set characteristics — tset, 356
  - set options — stty, 311
- test — condition command, 343
- test for Sun Workstation — sun, 316
- test machine is VAX — vax, 375
- text editing
  - ed — line editor, 111
  - edit — line editor, 123
  - ex — line editor, 123
  - sed — stream editor, 289
- text editor
  - vi — visual editor, 379
- text file
  - browse through — more, 215
  - browse through — page, 215
- text processing
  - awk — scan and process patterns, 20
  - cat — concatenate files, 31
  - search for patterns, 159
- text processing utilities
  - reverse lines in file — rev, 268
  - sort — sort or merge files, 303
  - spell — check spelling, 306
  - split — split file into pieces, 308
  - tail — display last part of file, 331
  - tr — translate characters, 352
  - tsort — topological sort, 360
  - ul — underline text, 362
  - uniq — remove repeated lines, 364
- tftpd — Trivial File Transfer Protocol server, 601
- time
  - display date and, 79
  - display in window, 46
- time — time command, 344
- time — time command — csh, 71
- timed — time server, 602
- timed event services
  - at — do job at specified time, 19
  - calendar — reminder service, 30
  - leave — remind you of leaving time, 181
- timed events — cron, 496
- times shell command — sh, 297
- tip — connect to remote system, 346
- topological sort — tsort, 360
- touch — update last modified date of file, 351
- tr — translate characters, 352
- translate and interpret Pascal programs — pc, 235
- translate characters — tr, 352
- transliterate protocol trace — trpt, 603
- trap shell command — sh, 297
- trek — Star Trek game, 452
- troff
  - checknr — check nroff/troff files, 40
  - col — filter reverse paper motions, 48
  - colcrt command, 49
  - remove constructs, 101
- troff — typeset documents, 353
- trpt — transliterate protocol trace, 603
- true — provide truth values, 355
- tset — set terminal characteristics, 356
- tsort — topological sort, 360
- tty — get terminal name, 361
- tune file system — tunefs, 604
- tunefs — tune file system, 604
- typeset documents — troff, 353

U

- ul — underline text, 362
- umask shell command — sh, 297
- umask — change/display file creation mask — csh, 71
- umount — unmount file system, 539
- unalias — remove aliases — csh, 71
- uncompress — uncompress files, 53
- uncompress files, 53
- underline text — ul, 362
- unexpand — spaces to tabs, 125
- unget — unget SCCS file, 363
- unhash — discard hash table — csh, 71
- uniq — remove repeated lines, 364
- units — convert units, 365
- UNIX to UNIX command execution — uux, 372
- UNIX to UNIX copy — uucp, 368
- unlimit — remove resource limitiations — csh, 71
- unmount file system — umount, 539
- unset — discard shell variables — csh, 72
- unsetenv — remove environment variables —

csh, 72
update — update super block, 605
update last modified date of file — touch, 351
update super block — sync, 329
uptime — display system up time, 366
USENET
checknews — check for news, 39
user
display effective name — whoami, 398
talk to another — talk, 332
write to another — write, 400
user id
substitute — su, 314
users
list last logins — last, 179
what are they doing — w, 389
who — who is logged in, 397
write to all — wall, 391
users — display users on system, 367
utilities
introduction, 1
uuclean — clean UUCP spool area, 606
UUCP
clean spool area — uuclean, 606
uucp — UNIX to UNIX copy, 368
UUCP log — uulog, 368
uudecode — decode binary file, 370
uuencode — encode binary file, 370
uulog — UUCP log, 368
uurec — receive news articles, 607
uusend — send file to remote host, 371
uux — UNIX to UNIX command execution, 372

V

val — validate SCCS file, 373
validate SCCS file — val, 373
variable substitution
in C-Shell, 62 *thru* 63
variables in C Shell, 72 *thru* 75
vax — test machine is VAX, 375
verification tools
lint — C program verifier, 183
plot graphics on — vplot, 383
version
identify, of file — what, 393
vfontinfo — examine font files, 376
vi — visual editor, 379
view — view file, 381
vipw — edit password file, 608
visual editor — vi, 379
vlp — format Lisp programs, 382
vmstat — display virtual memory statistics, 609
vplot — plot on Versatec, 383
vpq — display raster print queue, 384
vpr — print on raster printer/plotter, 384
vprint — print on raster printer/plotter, 384
vprm — remove jobs from raster print queue, 384
vswap — convert foreign font files, 386
vtroff — format document for raster printer, 387
vwidth — make font width table, 388

W

w — what are users doing, 389
wait shell command — sh, 297, 390
wait — wait for background process — csh, 72
wall — write to all users, 391
wc — count lines, words, characters in file, 392
what are users doing — w, 389
what — identify file version, 393
whatis — describe command, 394
whereis — find program, 395
which — find program file, 396
while shell command — sh, 293
while — repeat commands — csh, 72
who — who is logged in, 397
who is logged in on local network — rwho, 278
whoami — display effective user name, 398
whois — Internet directory service, 399
window
save context — lockscreen, 185
window environment — suntools, 318
window management
adjacentscreens command, 11
words in file
count — wc, 392
working directory
cd — change directory, 36
display name of — pwd, 253
worm — growing worm game, 453
worms — animate worms on display, 454
write — write to another user, 400
write EOF mark on magnetic tape — mt, 218
write to all users — wall, 391
write to all users on network — rwall, 277
wump — hunt the Wumpus game, 455

X

xget — receive secret mail, 402
xsend — send secret mail, 402
xstr — extract strings from C code, 403

Y

yacc — compiler generator, 404
yellow pages
make database — ypinit, 611
make dbm file — makedbm, 530
print values from database — ypcat, 406
rebuild database — ypmake, 612
yes — be repetitively affirmative, 405

ypcat — print values from YP database, 406
ypinit — make yellow pages database, 611
ypmake — rebuild yellow pages database, 612
yppoll — yellow pages administration, 614
yppull — yellow pages administration, 614
yppush — yellow pages administration, 614
ypserv — yellow pages server process, 615
ypwhich — who is yellow pages server, 617
yppasswdd — yellow pages password server, 613