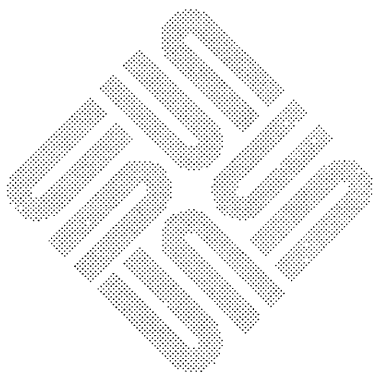
sun microsystems

Reference Manual Insertion Pages for Release 3.5

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

Part No: 800-2181-10
Revision A of 20 November 1987

Commands Reference Manual Insertion Pages



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 – make console text visible

SYNOPSIS

clear_colormap [**–no**] [**–f** *framebuffer*]

DESCRIPTION

Clear_colormap ensures that text displayed on the console is visible. If no options are specified it clears the frame buffer and initializes the first two colormap entries. If the frame buffer has an overlay plane it is also cleared and the overlay enable plane is set so that the entire overlay plane is displayed.

OPTIONS

–n Do not clear the frame buffer or overlay plane.

–o Do not clear the overlay plane or modify the overlay enable plane.

–f *framebuffer*

Operate on frame buffer device *framebuffer* instead of the default, /dev/fb.

NAME

clock, clocktool – display the time in a window

SYNOPSIS

clock [–s] [–t] [–r] [–d mdyaw] [–f]

DESCRIPTION

clock is a standard tool provided with the *SunView* environment.

clock displays the current time in its own window. In its open state, *clock* shows the date and time textual form. In its closed state, *clock* appears as a clock face which keeps time.

Note: In previous releases *clock* was known as *clocktool*. In the current release, *clocktool* is retained as a symbolic link to *clock*.

OPTIONS

–r causes *clock* 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.

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

–s start *clock* 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 Test mode — ignore the real time, and instead run in a loop continuously incrementing the time by one minute and displaying it.

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

When open, *clock* listens for keyboard input, toggling its state on four characters:

s or **S** toggles the display of seconds.

t or **T** toggles the 'test' mode.

SEE ALSO

suntools(1), date(1)

FILES

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

BUGS

If you reset the system time, *clock* will not reflect the new time until you change its state — open it if closed, close it if open. To reset the system time, see date(1).

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

The clock sometimes freezes. Bringing up the Frame Menu will unstick it.

NAME

cmdtool – Run a shell (or other program) from the SunView text facility

SYNOPSIS

cmdtool [**–C**] [**–P** *n*] [*program* [*args*]]

DESCRIPTION

cmdtool is a standard tool provided with the *SunView* environment.

When invoked, *cmdtool* runs a program (usually a shell) in a text-based command subwindow. Typed characters are inserted at the caret. If this program is a shell, it accepts commands and runs programs in the usual way, including commands that do cursor motion such as *vi*. (See *BUGS* below).

Text can be edited anywhere on the command line the same way as in any other text subwindow. Commands and their output are kept in a log which can be scrolled using the scrollbar, unless the command does cursor motion. The log file can also be edited, or even saved using the Save command in the text facility's pop-up menu. The Split command, also in the pop-up menu, can be used to create two or more independently scrolling views of the log.

DEFAULTS OPTIONS

/Tty/Append_only_log

TRUE is the standard default; it means that only the command line may be editted. *FALSE* permits editting of the entire log. See the descripton of *Enable Edit* below.

/Tty/Insert_makes_caret_visible

This entry determines how hard the command subwindow should try to keep the caret visible.

Same_as_for_text	Is the standard default; it means that the setting for **Insert_makes_caret_visible** will be taken from the **Text** category instead of **Tty** when a command subwindow is created.
If_auto_scroll	If the caret is showing, and an inserted newline would position it below the bottom of the screen as determined by **/Text/Lower_context,** the text is scrolled to keep it showing. The amount scrolled is controlled by **/Text/Auto_scroll_by.** See *textedit* (1) for more information.
Always	Upon any input action, if the caret is positioned off the screen, it is scrolled back into view.

/Tty/Checkpoint_frequency

0 is the standard default; it means that no checkpointing will take place. For a value *n* greater than zero, checkpointing will take place after every nth edit. Each character typed, each Get, and each Delete counts as an edit. At each checkpoint, an updated copy of the transcript is saved in a file whose name is constructed by appending two percent signs (%%) to the name of the transcript file. By default, the transcript file is named */tmp/tty.txt.nnnnnn*; in this case, the checkpoint file is named */tmp/tty.txt.nnnnnn%%*.

/Text/Edit_back_char

Set the character for deleting the character preceding the caret. Note: stty erase *has no effect*; text based tools only refer to the defaults database. The standard default is the DEL key.

/Text/Edit_back_word

Set the character for deleting the word preceding the caret. Note: stty werase *has no effect*; text based tools only refer to the defaults database. The standard default is CTRL-W.

/Text/Edit_back_line

Set the character for deleting from the newline preceding the caret to the caret. Note: stty kill *has no effect*; text based tools only refer to the defaults database. The standard default is CTRL-U.

COMMANDLINE OPTIONS

–C Redirect system Console output to this instance of the *cmdtool*. This will prevent system error messages from being printed in unexpected places on the screen. Moreover, since a *cmdtool* window is scrollable, console error messages that go off the top of the window can be scrolled back

for re-examination.

-P n Set the checkPoint frequency to n.

cmdtool also takes generic tool arguments; see *suntools* (1) for a list of these arguments.

program [*args*]
If a program argument is present, *cmdtool* executes it. Subsequent arguments will be assumed to be arguments of the program argument, and will be passed to it for execution. If there are no arguments, *cmdtool* runs the program corresponding to the SHELL environment variable. If this environment variable is not available, then *cmdtool* runs **/bin/sh**.

THE COMMAND SUBWINDOW

The subwindow of *cmdtool* is a command subwindow, which is also found in *dbxtool* and potentially in other tools as well. The command subwindow is based on the text facility. For more information about the text facility, see *Windows and Window-Based Tools: Beginner's Guide*. The pop-up menu associated with command subwindow is the same as that for the text facility (see *textedit* (1)), with two additional items, **Enable Edit** and **Disable Scrolling**.

Command subwindow now supports cursor motion. In order to do so, a new termcap type has been invented, sun-cmd. Command subwindow automatically sets the TERM environment variable to sun-cmd when it starts up. This means that if you rlogin to a machine that does not have sun-cmd defined in its termcap file, it will complain "Type sun-cmd unknown". To rectify this, "set TERM=sun". This also means that any program written using the curses package will automatically work in command subwindow, but a program written specifically for termcap type "sun" will not work. When allowing cursor motion, the command subwindow automatically takes on the appearance and characteristics of a tty subwindow (as in *shelltool*), with a wide margin where the scrollbar used to be. When the program doing cursor motion ends or goes to sleep, it automatically returns to its text subwindow appearance and behavior.

Programs that use CBREAK or RAW mode, or NO ECHO are now supported. Usually this support is invisible. However, rlogin and script have the unexpected characteristic of turning off the ability to edit the commandline with the mouse, although old-fashioned backspace, word and line kill still operate. This is because they go into RAW mode, demanding that cmdtool ship every character as it is typed, rather than echo locally.

THE COMMAND SUBWINDOW MENU

The generic text menu items will not be described here except for **Put, then Get**, as it approximates the functionality of **Stuff** in *shelltool* (1), and is also implemented for *shelltool*.

Put, then Get
When there is a selection, this item reads **Put, then Get**. It causes the selection to be copied both to the shelf and to the caret.

Put, then **Get**
When there is no selection but there is text on the shelf, **Put, then** is grayed out, though **Get** remains active. Selecting this item causes the contents of the shelf to be copied to the caret. When there is no selection and nothing is on the shelf, this item is inactive.

Enable Edit
If the defaults entry **Append_only_log** is set to *TRUE*, but at some point you want to edit the log, selecting this menu item makes editting the log possible. When the log is editable, this item reads **Disable Edit**, and selecting it makes the log read-only before the start of the command line.

Disable Scrolling
Normally, command subwindow enters tty subwindow mode automatically when you start up a program that does cursor motion. If it fails to do so, perhaps because the program does not use curses and does not interpret termcap correctly, or if you want tty subwindow mode because it supports something that command subwindow doesn't (see *BUGS* below), this menu item lets you enter tty subwindow mode manually. The tty subwindow that comes up has **Enable Scrolling** added to its menu to allow manual return to text subwindow mode. Disable and Enable Scrolling

correspond to the termcap "ti" and "te" escape sequences for the "sun-cmd" termcap entry, respectively. If you manually disable and enable scrolling often, you may find it useful to assign these sequences to function keys in your .ttyswrc file. Refer to /etc/termcap for the escape sequences and refer to *shelltool* (1) for a description of .ttyswrc.

Certain text facility accelerators that are especially useful in command subwindows are described here. See *textedit* (1) for more information.

CTRL-RETURN

Holding down the control key while typing newline (carriage return) positions the caret at the bottom and scrolls it into view, as determined by the defaults option **/Text/Lower_context**.

CTRL-P is an accelerator for the **Put, then Get** menu item described above.

CAPS-lock

Bound to F1, it causes subsequent keyboard input to be uppercase. This key is a toggle; striking it a second time undoes the effect of the first strike.

FILES

/tmp/tty.txt.<*pid*> (transcript file)

~/.textswrc

~/.ttyswrc

SEE ALSO

shelltool(1), suntools(1), textedit(1), defaultsedit(1),

Windows and Window-Based Tools: Beginner's Guide

BUGS

Full terminal emulation is not complete. Some manifestations of this deficiency are:

- File completion in the C-shell does not work.
- Emphasized text (eg. in man pages) does not show up as emphasized.

Occasionally the C-shell exits due to a hangup signal after the first command entered by the user. This causes cmdtool to exit as well, and print a message to either the console or the window it was started from. The message reads:
A command window has exited because its child exited.
Its child's process id was <*pid*> and it died due to signal 1.

Compression: *xx.xx%*
Percentage of the input saved by compression. (Relevant only for –v.)

-- not a regular file: unchanged
When the input file is not a regular file, (e.g. a directory), it is left unaltered.

-- has *xx* other links: unchanged
The input file has links; it is left unchanged. See *ln*(1) for more information.

-- file unchanged
No savings are achieved by compression. The input remains uncompressed.

SEE ALSO

A Technique for High Performance Data Compression , Terry A. Welch, *IEEE Computer*, vol. 17, no. 6 (June 1984), pp. 8-19.

compact(1), pack(1)

BUGS

Although compressed files are compatible between machines with large memory, –**b**12 should be used for file transfer to architectures with a small process data space (64KB or less).

compress should be more flexible about the existence of the **.Z** suffix.

NAME

cp – copy files

SYNOPSIS

cp [–i] [–p] [–rR] file1 file2

cp [–i] [–p] [–rR] 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.

–p Preserve: attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the present *umask*.

–r

–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. In the case of a symbolic link, the link itself is not replicated. Instead, *cp* duplicates the contents of the file pointed to by the symbolic link.

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.

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(1V)

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**. Invalid combinations silently ignore all but the last mutually-exclusive keyword.

NAME

defaultsedit, defaults_merge, defaults_from_input defaults_to_indentpro, defaults_to_mailrc, indentpro_to_defaults, lockscreen_default, mailrc_to_defaults, scrolldefaults – window- and mouse-based default parameters editor

SYNOPSIS

defaultsedit

DESCRIPTION

defaultsedit is a standard tool provided with the *SunView* environment.

defaultsedit presents a convenient user interface for inspecting and setting default parameters. It can be viewed as a replacement for the traditional UNIX *defaultsedit* to manipulate options to the programs *indent*, *mail* and *mailtool*, *stty*, and *defaultsedit*, as well as the *menu*, *scrollbar*, *text subwindow* and *tty subwindow* packages and the *SunView* environment.

Any program or package which a user can customize by setting or changing a parameter could be written such that it gets its options from the database manipulated through *defaultsedit*. For information on how to do this see the chapter on the Defaults Database in the *SunView System Programmer's Guide*.

OPTIONS

defaultsedit accepts all of the generic tool arguments discussed in suntools(1).

SUBWINDOWS

defaultsedit consists of four subwindows. From top to bottom they are:

control — contains the name of the category currently displayed, and buttons labeled SAVE, QUIT, RESET, and EDIT ITEM. To change the category, click on the word CATEGORY with the left mouse button, or use the menu that pops up when you click with the right mouse button.

message — a small text subwindow where messages from *defaultsedit* are displayed.

parameters — shows all current default parameter names with corresponding values. Clicking the left mouse button over a parameter displays a help string in the message subwindow.

edit — a small text subwindow which enables text editing of parameter values. This is useful for very long text values, such as a long mailing list.

USING DEFAULTSEDIT

SAVE — Saves the current values for all categories in your private database — that is, the *.defaults* file in your home directory.

QUIT — exits without saving any changes.

RESET — resets the default parameters of the current category to the values in your private database. This is useful if you change some values, then change your mind and want to restore the original values.

EDIT ITEM — Pressing the right mouse button over the EDIT ITEM button brings up a menu with three choices: COPY ITEM, DELETE ITEM and EDIT LABEL. Only text or numeric items can be edited. Also, note that edits made using this menu will appear only in your private defaults database, not in the master database. The three editing operations are described below.

COPY ITEM — Selecting COPY ITEM causes the current item to be duplicated. You can then edit both the label and the value of the the newly created item. Only items with text or numeric values can be copied in this way. COPY ITEM is useful when you want to change the number of instances of a certain type of item — for example, to insert a new mail alias into your defaults database.

DELETE ITEM — Selecting DELETE ITEM will delete the current item from your private database. It cannot be permanently deleted if the corresponding node is present in the master database.

However, you can make it behave like an undefined node by giving it the special value *\255Undefined\255*.

EDIT LABEL

Selecting EDIT LABEL allows you to edit the label of the current item. When you select EDIT LABEL, the label of the current item changes from bold to normal face. Then you can select the label and edit it as a normal panel text item.

ENVIRONMENT

DEFAULTS_FILE

The value of this environment variable indicates the file from which SunView defaults are read. When it is undefined, defaults are read from the *.defaults* file in your home directory.

FILES

~/.defaults /usr/lib/defaults/*.d

Note: A performance optimzation may be enabled by setting the Private_only parameter in the Defaults category. If this is set to True, only the user's private defaults file is consulted.

SEE ALSO

Windows and Window-Based Tools: Beginner's Guide

The SunView System Programmer's Guide

BUGS

Editing of choice items or categories is not supported by *defaultsedit*. Neither is editing of the master defaults database — to add a new program to the master defaults database, you have to edit a master defaults textfile.

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.

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 '.h'.

–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, e – text editor

SYNOPSIS

ex [–] [**–R**] [**–r**] [**–t** *tag*] [*+command*] [**–v**] [**–x**] [**–w***nnn*] [**–l**] *file* ...
edit [*options*]

DESCRIPTION

ex, a line editor, is the root of a family of editors that includes *edit*, *ex*, and *vi* (the display editor). In most cases *vi* is preferred for interactive use.

OPTIONS

– supress all interactive feedback to the user — useful for processing *ex* scripts in shell files.

–R Read only. Do not overwrite the original file.

–r recover the indicated *file*s after a system crash.

–t *tag* edit the file containing the tag *tag*. A tags database must first be created 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.

ENVIRONMENT

The editor recognizes the environment variable EXINIT as a command (or list of commands separated by | characters) to run when it starts up. If this variable is undefined, the editor checks for startup commands in the file *~/.exrc* file, which you must own. However, if there is a *.exrc* owned by you in the current directory, the editor takes its startup commands from this file—overriding both the file in your home directory and the environment variable.

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

NAME

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [–tabstop] [*–tab1,tab2,...,tabn*] [*filename* ...]
unexpand [**–a**] [*filename* ...]

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.

NAME

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

SYNOPSIS

grep [**–v**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] [**–h**] [**–w**] [**–e**] *expression* [*file* ...]

egrep [**–v**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] [**–h**] [**–e** *expression*] [**–f** *file*]
[*expression*] [*file* ...]

fgrep [**–v**] [**–x**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] [**–h**]
[**–e** *string*] [**–f** *file*] [*string*] [*file* ...]

SYSTEM V SYNOPSIS

grep [**–v**] [**–c**] [**–l**] [**–n**] [**–b**] [**–i**] [**–s**] *expression* [*file* ...]

DESCRIPTION

Commands of the *grep* family search the input *file*s (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* 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. It is safest to enclose the entire *expression* argument in single quotes ´...´.

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 rather than displaying the lines which match.

–l List only 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.

–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 \>. *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 a list of strings separated by newlines (*fgrep*) from *file*.

SYSTEM V OPTIONS

The System V version of *grep* does not recognize the **–h**, **–w**, or **–e** options. The **–s** option indicates that error messages for nonexistent or unreadable files should be suppressed, not that all messages should be suppressed.

REGULAR EXPRESSIONS

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

c An ordinary character (*not* one of the special characters discussed below) is a one-character regular expression that matches that character.

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

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([ ]).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* regular expression, or when it immediately follows the left of a pair of square brackets ([ ]).

c. $ (currency symbol), which is special at the *end* of an entire regular expression.

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

[*string*]

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

The following rules may be used to construct regular expressions:

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

\(A regular expression enclosed between the character sequences \(and \) matches whatever the unadorned regular expression matches. (*grep* only).

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

concatenation

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

\< The sequence \< in a regular expression constrains the one-character regular expression immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> The sequence \> in a regular expression constrains the one-character regular expression immediately following it only to match something at the end of a "word"; that is, either at the end of a line, or just before a character which is neither a letter, digit, nor underline.

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

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

The construction ^*entire regular expression*$ constrains the entire regular expression to match the entire line.

egrep accepts regular expressions of the same sort *grep* does, except for \(, \), *n*, \<, and \>, with the addition of:

* A regular expression (not just a one-character regular expression) followed by an asterisk (*) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

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

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

| Alternation: two regular expressions separated by | or newline match either a match for the first or a match for the second.

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

The order of precedence of operators at the same parenthesis level is [ ] (character classes), then * + ? (closures), then concatenation, then | (alternation) and newline.

SYSTEM V REGULAR EXPRESSIONS

The System V version of *grep* does not accept \< or \> in a regular expression, and accepts the following additional item in a regular expression:

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

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(1V)	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

For */bin/grep* the order in which concatenated options appear makes a difference in the resulting output.

Lines are limited to 1024 characters by *grep*; longer lines are truncated.
If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.
The combination of –l and –v options does *not* produce a list of files in which a regular expression is not found. To get such a list, use the C-Shell construct:

```
foreach file (*)                    if ('grep "re" $file | wc -l' == 0) echo $file
end
```

Ideally there should be only one *grep*.

Fill (Button) Fill canvas with current rectangular fill pattern.

Invert (Button) Invert each pixel represented on the canvas.

Paintbrush

Select from among five painting modes. 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 available fonts or press and hold the right mouse button to bring up a menu of fonts.

Load This is the rasterop to be used when loading a file in from disk. (See the *Pixrect Reference Manual* for 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/iconedit

NAME

id – print user and group IDs and names

SYNOPSIS

/usr/5bin/id

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install this command.

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

SEE ALSO

getuid(2)

NAME

indent – indent and format C program source

SYNOPSIS

indent [*input-file* [*output-file*]] [**–bacc** | **–nbacc**] [**–bad** | **–nbad**] [**–bap** | **–nbap**] [**–bbb** | **–nbbb**] [**–bc** | **–nbc**] [**–bl**] [**–br**] [**–bs** | **–nbs**] [**–c***n*] [**–cd***n*] [**–cdb** | **–ncdb**] [**–ce** | **–nce**] [**–ci***n*] [**–cli***n*] [**–d***n*] [**–di***n*] [**–eei** | **–neei**] [**–fc1** | **–nfc1**] [**–i***n*] [**–ip** | **–nip**] [**–l***n*] [**–lc***n*] [**–lp** | **–nlp**] [**–pcs** | **–npcs**] [**–npro**] [**–psl** | **–npsl**] [**–sc** | **–nsc**] [**–sob** | **–nsob**] [**–st**] [**–troff**] [**–v** | **–nv**]

DESCRIPTION

Indent is a C program formatter. It reformats the C program in the *input-file* according to the switches. The switches which can be specified are described below. They may appear before or after the file names.

NOTE: If you only specify 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 '/blah/blah/file', the backup file is named file.*BAK*.

If *output-file* is specified, *indent* checks to make sure it is different from *input-file*.

OPTIONS

The options listed below control the formatting style imposed by *indent*.

–bap,–nbap If **–bap** is specified, a blank line is forced after every procedure body. Default: **–nbap.**

–bacc,–nbacc If **–bacc** is specified, a blank line is forced around every conditional compilation block. ie. in front of every #ifdef and after every #endif. Other blanklines surrounding these will be swallowed. Default: **–nbacc.**

–bad,–nbad If **–bad** is specified, a blank line is forced after every block of declarations. Default: **–nbad.**

–bbb,–nbbb If **–bbb** is specified, a blank line is forced before every block comment. Default: **–nbbb.**

–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.**

–br,–bl Specifying **–bl** lines up compound statements like this:

```
if (...)
{
    code
}
```

Specifying **–br** (the default) makes them look like this:

```
if (...) {
    code
}
```

–bs,–nbs Enables (disables) the forcing of a blank after **sizeof.** Some people believe that **sizeof** should appear as though it were a procedure call (**–nbs**, the default) and some people believe that since **sizeof** is an operator, it should always be treated that way and should always have a blank after it.

–c*n* The column in which comments on code start. The default is 33.

–cd*n* The column in which comments on declarations start. The default is for these comments to start in the same column as those on code.

–cdb,–ncdb Enables (disables) the placement of comment delimiters on blank lines. With this option enabled, comments look like this:

```
/*
 * this is a comment
 */
```

Rather than like this:

```
/* this is a comment */
```

This only affects block comments, not comments to the right of code. The default is **–cdb .**

–ce,–nce Enables (disables) forcing 'else's to cuddle up to the immediatly preceeding '}'. The default is **–ce .**

–ci*n* Sets the continuation indent to be *n*. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless **–lp** is in effect. **–ci** defaults to the same value as **–i.**

–cli*n* Causes case labels to be indented *n* tab stops to the right of the containing **switch** statement. **-cli0.5** causes case labels to be indented half a tab stop. The default is **–cli0 .**

–d*n* Controls the placement of comments which are not to the right of code. The default **–d1** means that such comments are placed one indentation level to the left of code. Specifying **–d0** lines up these comments with the code. See the section on comment indentation below.

–di*n* Specifies the indentation, in character positions, from a declaration keyword to the following identifier. The default is **–di16 .**

–eei,–neei If **–eei** is specified, and extra expression indent is applied on continuation lines of the expression part of if() and while(). These continuation lines will be indented one extra level – twice instead of just once. This is to avoid the confusion between the continued expression and the statement that follows the if() or while(). Default: **–neei.**

–fc1,–nfc1 Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading '/' is in column 1 have been carefully hand formatted by the programmer. In such cases, **–nfc1** should be used. The default is **–fc1.**

–i*n* The number of spaces for one indentation level. The default is 4.

–ip,–nip Enables (disables) the indentation of parameter declarations from the left margin. The default is **–ip .**

–l*n* Maximum length of an output line. The default is 75.

–lc*n* Sets the line length for block comments to *n*. It defaults to being the same as the usual line length as specified with **–l.**

–lp,–nlp Lines up code surrounded by parenthesis in continuation lines. If a line has a left paren which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left paren. For example, here is how a piece of continued code looks with -nlp in effect:

```
p1 = first_procedure(second_procedure(p2, p3),
    third_procedure(p4, p5));
```

With **-lp** in effect (the default) the code looks somewhat clearer:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

Inserting a couple more newlines we get:

```
p1 = first_procedure(second_procedure(p2,
                                      p3),
                     third_procedure(p4,
                                     p5));
```

–npro Causes the profile files, './.indent.pro' and '~/.indent.pro', to be ignored.

–pcs , –npcs If true (**-pcs**) all procedure calls will have a space inserted between the name and the '('. The default is **–npcs**

–psl , –npsl If true (**-psl**) the names of procedures being defined are placed in column 1 – their types, if any, will be left on the previous lines. The default is **-psl**

–sc,–nsc Enables (disables) the placement of asterisks ('*'s) at the left edge of all comments.

–sob,–nsob If **–sob** is specified, indent will swallow optional blank lines. You can use this to get rid of blank lines after declarations. Default: **–nsob**

–st Causes **indent** to take its input from stdin, and put its output to stdout.

–T*typename* Adds *typename* to the list of type keywords. Names accumulate: **–T** can be specified more than once. You need to specify all the typenames that appear in your program that are defined by **typedef**s – nothing will be harmed if you miss a few, but the program won't be formatted as nicely as it should. This sounds like a painful thing to have to do, but it's really a symptom of a problem in C: **typedef** causes a syntactic change in the laguage and *indent* can't find all **typedef**s.

–troff Causes **indent** to format the program for processing by troff. It will produce a fancy listing in much the same spirit as **vgrind.** If the output file is not specified, the default is standard output, rather than formatting in place.

The usual way to get a troff'd listing is with the command

```
indent -troff program.c | troff -mindent
```

–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.**

FURTHER DESCRIPTION

You may set up your own 'profile' of defaults to *indent* by creating a file called *.indent.pro* in either your login directory or the current directory and including whatever switches you like. A '.indent.pro' in the current directory takes precedence over the one in your login directory. 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 switches should be separated by spaces, tabs or newlines.

Comments

'Box' comments. *Indent* assumes that any comment with a dash or star immediately after the start of comment (that is, '/*–' or '/**') is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that its indentation may be adjusted to account for the change in indentation of the first line of the comment.

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. Blank lines break paragraphs.

Comment indentation

If a comment is on a line with code it is started in the 'comment column', which is set by the **–c***n* command line parameter. Otherwise, the comment is started at *n* indentation levels less than where code is currently being placed, where *n* is specified by the **–d***n* command line parameter. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

Preprocessor lines

In general, *indent* leaves preprocessor lines alone. The only reformmatting that it will do is to straighten up trailing comments. It leaves imbedded comments alone. Conditional compilation (**#ifdef...#endif**) is recognized and *indent* attempts to correctly compensate for the syntactic peculiarites introduced.

C syntax

Indent understands a substantial amount about the syntax of C, but it has a 'forgiving' parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like:

```
#define forever for(;;)
```

is handled properly.

FILES

./.indent.pro	profile file
~/.indent.pro	profile file
/usr/lib/tmac/tmac.indent	Troff macro package for "indent -troff" output.

BUGS

Indent has even more switches than *ls*.

A common mistake that often causes grief is typing:

```
indent *.c
```

to the shell in an attempt to indent all the C programs in a directory. This is probably a bug, not a feature.

The **-bs** option splits an excessivly fine hair.

NAME

lockscreen, lockscreen_default – maintain window context, prevent unauthorized access and reduce phosphor burn.

SYNOPSIS

lockscreen [**–b** *program*] [**–e**] [**–n**] [**–r**] [**–t** *seconds*] [*gfx-program*] [*gfx-program-args*]

DESCRIPTION

Lockscreen is a standard tool provided under the SunView environment that preserves the current state of the display while the machine is not in use. When run, the display is cleared to black and the *gfx-program* is run. It is assumed that the *gfx-program* will provide moving graphics to limit phosphor burn of the video display that might otherwise occur from leaving the same static window configuration displayed for a long time. If no *gfx-program* is provided, a suitable default program is run.

Lockscreen prevents unauthorized access by requiring the user's password before restoring the window context. When any keyboard or mouse button is pressed, the graphics screen is replaced by a password screen that displays the user name, a small box with a bouncing logo, 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 context is immediately restored.

When the password screen appears:

1) Restore the window context by entering the user's password followed by a carriage return (this password is not echoed on the screen) or,

2) Point to the black box and click the left button to return to the graphics display.

If neither of the above actions is taken, *gfx_program* will resume execution after the interval specified with the **–t** option, as described below.

OPTIONS

–b *program*
: Allow an additional program to be run as a child process of *lockscreen*. This background process could be a compile server or some other useful program that the user wants run while lockscreen is running. No arguments are passed to this program.

–e
: Add the *Exit Desktop* choice to the password screen. If pointed to and clicked, the SunView environment is exited and the current user is logged out.

–n
: Require no password to reenter the window environment.

–r
: Allow the use of the user name **root** in the **Name:** field of the password screen. Normally, **root** is not accepted as a valid user name.

–t *seconds*
: After *seconds* seconds, clear the password screen and restart the gfx-program. The default is 5 minutes (300 seconds).

[*gfx-program*] [*gfx-program-args*]
: Run this program after clearing the screen to black. If no *program* argument is present, *lockscreen* will try to run *lockscreen_default* if it exists on the standard search path, otherwise a bouncing Sun logo will appear. If *gfx-program-args* are specified and the *gfx-program* isn't then the args are passed to *lockscreen_default*. *Lockscreen_default* is typically a life program displaying the successive generations. *Lockscreen* will not search for *lockscreen_default* if the *gfx-program* is specified explicitly as "".

FILES

/usr/bin/lockscreen_default
: The default *gfx-program*. If a file named *lockscreen_default* appears earlier in the search path, that file is used instead.

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 userid to another.

When used with no argument, *login* requests a user name and password (if appropriate). Echoing is turned off (if possible) while typing the password.

When successful, *login* updates accounting files, informs you of the existence of any mail, prints the message of the day, and displays the time you last logged in (unless you have a *.hushlogin* file in your home directory — mainly used by nonhuman users, such as *uucp*).

login initializes the user and group IDs and the working directory, then starts a command interpreter shell (usually either */bin/sh* or */bin/csh* according to specifications found in the file */etc/passwd*. (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 with information specifying home directory, command interpreter, terminal-type (if available) and username.

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. If the file */etc/securetty* exists, only those terminals listed in that file provide login access to the super-user **root**. For example, if the file contained:

console

The super-user could only log in on the console.

The *login* command, recognized by *sh* and *csh*, is executed directly (without forking), and terminates that shell. To resume working, you must log in again.

login times out and exits if its prompt for input is not answered within a reasonable time.

When the Bourne shell (*sh*) starts up, it reads a file called *.profile* from your home directory (that of the username you use to log in). When the C-Shell (*csh*) starts up, it reads a file called *.cshrc* from your home directory, and then reads a file called *.login*.

The shells read these files only if they are owned by the person logging in.

FILES

/usr/adm/lastlog	time of last login
/usr/adm/wtmp	accounting
*/usr/spool/mail/**	mail
/usr/ttytype	terminal types
/usr/ucb/quota	quota check
~/.hushlogin	makes login quieter
/etc/motd	message-of-the-day
/etc/nologin	stop login, print message
/etc/passwd	password file
/etc/securetty	terminals allowing the super-user to log in
/etc/utmp	accounting

SEE ALSO

init(8), getty(8), mail(1), passwd(1), passwd(5), environ(5V), shutdown(8), utmp(5)

DIAGNOSTICS

"Login incorrect," if the name or the password is bad (or mistyped).
"No Shell", "cannot open password file", "no directory": ask your system administrator for assistance.

NAME

lpr – send job to printer

SYNOPSIS

lpr [**–P***printer*] [**–#***num*] [**–C***class*] [**–J***job*] [**–T***title*] [**–i** [*num*]] [**–1234***font*]
[**–w***num*] [**–B**] [**–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*

Send output to the named *printer*. Otherwise send output to the printer named in the PRINTER environment variable, or to the default printer, **lp**. If there is no entry in */etc/printcap* for **lp** , *lpr* supplies a default set of printer capabilities.

–#*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*) 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*.

–i[*num*] Indent output *num* spaces. If *num* is not given, eight spaces are used as default.

–1*font*
–2*font*
–3*font*
–4*font* Mount the specified *font* on font position **1**, **2**, **3** or **4**. The daemon will construct a *.railmag* file in the spool directory that indicates the mount by referencing */usr/lib/vfont/font*.

–w*num* Use *num* as the page width for *pr*.

–r Remove the file upon completion of spooling. **–B** Omit page headers.

–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 on the local host (files sent to remote printer hosts are copied anyway), and only with named 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* to format the files (**lpr –p** is very much like **pr | lpr**).
–l Print control characters and suppress page breaks.
–t The files contain *troff* (cat phototypesetter) binary data.

–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, see *rasterfile*(5).
–c This option currently is unassigned.
–f Interpret the first character of each line as a standard FORTRAN carriage control character.

If no *filter-option* is given, '**%!**' as the first two characters indicates that the file contains Postscript commands.

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(1V), printcap(5), lpc(8), lpd(8), rasterfile(5), screendump(1)

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% **/usr/etc/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.

BUGS

lpr –p is not equivalent to **pr | lpr**. **lpr –p** puts the current date at the top of each page, rather than the date last modified, and inserts a header page between each file printed.

The **–p** and **–#** options don't work well together; the second and subsequent copies do not include the file name in each page's title.

Fonts for *troff* and *tex* reside on the host with the printer. It is currently not possible to use local font libraries.

SEE ALSO

adb(1), dbxtool(1), dbx(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

on – execute a command remotely

SYNOPSIS

on [**–i**] [**–n**] [**–d**] *host command* [*argument*] ...

DESCRIPTION

The *on* program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

Standard input is connected to standard input of the remote command, and standard output and standard error from the remote command are sent to the corresponding files for the *on* command.

OPTIONS

–i Interactive mode: use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.

–n No Input: this option causes the remote program to get end-of-file when it reads from standard input, instead of passing standard input from the standard input of the *on* program. For example, **–n** is necessary when running commands in the background with job control.

–d Debug mode: print out some messages as work is being done.

SEE ALSO

rexd(8), exports(5)

DIAGNOSTICS

unknown host	Host name not found
cannot connect to server	Host down or not running the server
can't find .	Problem finding the working directory
can't locate mount point	Problem finding current file system

Other error messages may be passed back from the server.

BUGS

The SunView window system can get confused by the environment variables.

When the working directory is remote mounted over NFS, a ^Z hangs the window.

NAME

overview – run a program from SunView that takes over the screen

SYNOPSIS

overview [**–w**] [*generic_tool_flags*] *program_name* [*arguments*] ...

DESCRIPTION

Bitmap graphics based programs that are not SunView based can be run from SunView using *overview*. *Overview* shows an icon in SunView when *overview* is brought up iconic (**–Wi** flag) or when the program being run by *overview* is suspended (for example using ctrl-Z). Opening the *overview* icon, or starting *overview* non-iconic, starts the program named on the command line. *Overview* supresses SunView so that SunView window applications won't interfere with the program's display output or input devices.

Overview runs programs that fit the following profile:

own display	The program needs to own the bits on the screen. It doesn't use the *sunwindow* or *suntool* libraries to arbitrate the use of the display and input devices between processes.
keyboard input from stdin	The program takes keyboard input from *stdin* directly.
mouse input from /dev/mouse	The program takes locator input from the mouse directly.

OPTIONS

–w This flag is used to specify that the program being run creates its own SunWindows window in order to receive the serialized input stream from the keyboard and mouse that is provided by the SunWindows kernel driver. **–w** tells *overview* to not convert SunWindows input into ASCII which is then sent to the program being run under *overview* via a pty. *X* and *NeWS* are programs that fall in this category (as of Dec 86, which is subject to change in the future).

SEE ALSO

Windows and Window-Based Tools: Beginner's Guide

BUGS

Users of *overview* on a Sun-3/110 frame buffer multiple frames should be aware of the existence of plane groups for pre-3.2 applications. You can't successfully run pre-3.2 applications under *overview* if *overview* itself is running in the color buffer. If you start *overview* so that it is not running in the overlay plane, then the enable plane isn't be properly set up for viewing the application. This means that you can't run *overview* with the **-Wf** or **-Wb** generic tool arguments. Also, you can't run *overview* on a desktop created by *suntools* using the **-8bit_color_only** option.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [–**f**] *filename* ...

pcat *filename* ...

unpack *filename* ...

DESCRIPTION

pack attempts to store the specified files in a packed form using Huffman (minimum redundancy) codes on a byte-by-byte basis. Wherever possible (and useful), each input file *filename* is replaced by a packed file *filename*.z with the same access modes, access and modified dates, and owner as those of *filename*. If *pack* is successful, *filename* will be removed.

Packed files can be restored to their original form using *unpack* or *pcat*.

The amount of compression obtained depends on the size of the input file and the frequency distribution of its characters.

Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks unless the distribution of characters is very skewed. This may occur with printer plots or pictures.

Typically, large text-files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression. Their packed versions come in at about 90% of the original size.

No packing will occur if:

the file appears to be already packed
the file name has more than 12 characters
the file has links
the file is a directory
the file cannot be opened
no disk storage blocks will be saved by packing
a file called *name*.z already exists
the .z file cannot be created
an I/O error occurred during processing

The last segment of the filename must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be packed.

pcat does for packed files what *cat*(1V) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. To view a packed file named *name*.z use:

pcat *filename*.z

or just:

pcat *filename*

To make an unpacked copy without destroying the packed version, use

pcat *filename*> *newname*

Failure may occur if:

the filename (exclusive of the .z) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.z (or just *name*, if *name* ends in .z). If this file appears to be a packed, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the "unpacked" name already exists
the unpacked file cannot be created.

OPTIONS

– Print compression statistics for the following filename or names on the standard output. Subsequent –'s between filenames toggle statistics off and on.

–f Force packing of *filename*. This is useful for causing an entire directory to be packed, even if some of the files will not benefit.

DIAGNOSTICS

pack returns the number of files that it failed to compress.

pcat returns the number of files it was unable to unpack.

unpack returns the number of files it was unable to unpack.

SEE ALSO

cat(1), compact(1)

–b Causes the printing of the body of the SCCS file.

–e This keyletter implies the **d**, **i**, **u**, **f**, and **t** keyletters and is provided for convenience.

–y[*SID*] This keyletter will cause the printing of the delta table entries to stop when the delta just printed has the specified SID. If no delta in the table has the specified SID, the entire table is printed. If no SID is specified, the first delta in the delta table is printed. This keyletter will cause the entire delta table entry for each delta to be printed as a single line (the newlines in the normal multi-line format of the **d** keyletter are replaced by blanks) preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

–c[*cutoff*]
This keyletter will cause the printing of the delta table entries to stop if the delta about to be printed is older than the specified cutoff date-time (see *get*(1) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

–r[*rev-cutoff*]
This keyletter will cause the printing of the delta table entries to begin when the delta about to be printed is older than or equal to the specified cutoff date-time (see *get*(1) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. (In this case, nothing will be printed). As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

If any keyletter but **y**, **c**, or **r** is supplied, the name of the file being processed (preceded by one newline and followed by two newlines) is printed before its contents.

If none of the **u**, **f**, **t**, or **b** keyletters is supplied, the **d** keyletter is assumed.

Note that the **s** and **i** keyletters, and the **c** and **r** keyletters are mutually exclusive; therefore, they may not be specified together on the same *prt* command.

The form of the delta table as produced by the **y**, **c**, and **r** keyletters makes it easy to sort multiple delta tables by time order. For example, the following will print the delta tables of all SCCS files in directory *sccs* in reverse chronological order:

```
prt –c sccs | grep . | sort ´–rttab´ +2 –3
```

When both the **y** and **c** or the **y** and **r** keyletters are supplied, *prt* will stop printing when the first of the two conditions is met.

SEE ALSO

sccs(1), admin(1), get(1), delta(1), prs(1), what(1), help(1), sccsfile(5)

Programming Utilities for the Sun Workstation.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – process status

SYNOPSIS

ps [**acCegklsStuvwx**] [*num*] [*kernel_name*] [*c_dump_file*] [*swap_file*]

DESCRIPTION

ps displays information about processes. Normally, only those processes that are started by you and are attached to a controlling terminal (see *termio*(4V)) are shown. Additional categories of processes can be added to the display using various options. In particular, the **a** option allows you to include processes that are not owned by you (that do not have your user ID), and the **x** option allows you to include processes without control terminals. When you specify both **a** and **x**, you get processes owned by anyone, with or without a control terminal. *ps* displays the process id, under PID; the control terminal (if any), under TT; the cpu time used by the process so far, including both user and system time), under CPU; the state of the process, under STAT; and finally, an indication of the COMMAND that 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; 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 Include information about processes owned by others.

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 terminal 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 terminal, 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 Include processes with no controlling terminal.

num 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 kilobyte units). With the **u** option, values shown include the size of the text segment. With the **v** option, values shown do not include the text segment.
RSS	real memory (resident set) size of the process (in kilobyte 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 page fram number or swap area position
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 terminal names

SEE ALSO

kill(1), w(1), pstat(8), termio(4V)

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to the current state.

NAME

ranlib – convert archives to random libraries

SYNOPSIS

ranlib [**–t**] *archive* ...

DESCRIPTION

ranlib converts each *archive* to a form that can be linked 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. Sufficient temporary file space must be available in the file system that contains the current directory.

OPTIONS

–t option, *ranlib* only "touches" the archives and does not modify them. This is useful after copying an archive or using the **–t** option of *make*(1) in order to avoid having *ld*(1) complain about an "out of date" symbol table.

SEE ALSO

ld(1), ar(1), lorder(1), make(1)

BUGS

Because generation of a library by *ar* and randomization of the library by *ranlib* are separate processes, phase errors are possible. The linker, *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

rasfilter8to1 – convert an 8-bit deep rasterfile to a 1-bit deep rasterfile

SYNOPSIS

rasfilter8to1 [**–d**] [**–rgba** *threshold*] [*infile* [*outfile*]]

DESCRIPTION

Rasfilter8to1 reads the 8-bit deep rasterfile *infile* (standard input default) and converts it to the 1-bit deep rasterfile *outfile* (standard output default) by thresholding or ordered dither. The output format is Sun standard rasterfile format (see */usr/include/rasterfile.h*). This command is useful for viewing 8-bit rasterfiles on devices that can only display monochrome images.

OPTIONS

–d Use ordered dither to convert the input file instead of thresholding.

–rgba *threshold*

Set the threshold for the red, green, blue, and average pixel color values. Pixels whose color values are greater than or equal to all of the thresholds are given a value of 0 (white) in the output rasterfile; other pixels are set to 1 (black). The average threshold defaults to 128, the individual thresholds to zero.

EXAMPLE

The command

tutorial% screendump -f /dev/cgtwo0 | rasfilter8to1 | lpr -Pversatec -v

prints a monochromatic representation of the */dev/cgtwo0* frame buffer on the printer named "versatec" using the "v" output filter (see /etc/printcap).

FILES

/usr/lib/rasfilters/* Filters for non-standard rasterfile formats

SEE ALSO

lpr(1), rastrepl(1), screendump(1), screenload(1)

File I/O Facilities for Pixrects in *Pixrect Reference Manual*

NAME

rastrepl – magnify a raster image by a factor of two

SYNOPSIS

rastrepl [*infile* [*outfile*]]]]

DESCRIPTION

Rastrepl reads the rasterfile *infile* (standard input default) and converts it to the rasterfile *outfile* (standard output default) which is twice as large in width and height. Pixel replication is used to magnify the image. The output file has the same type as the input file.

EXAMPLES

tutorial% **screendump | rastrepl | lpr –Pversatec –v**

sends a rasterfile containing the current frame buffer contents to the Versatec plotter, doubling the size of the image so that it fills a single page.

FILES

/usr/lib/rasfilters/* Filters for non-standard rasterfile formats

SEE ALSO

lpr(1), screendump(1), screenload(1)

File I/O Facilities for Pixrects in *Pixrect Reference Manual*

NAME

ratfor – rational FORTRAN dialect

SYNOPSIS

ratfor [**–6***c*] [**–C**] [**–h**] [filename ...]

DESCRIPTION

ratfor converts the rational FORTRAN dialect into ordinary FORTRAN 77. It provides control flow constructs essentially identical to those in C. See the *FORTRAN 77 Programmer's Guide* for a description of the Ratfor language.

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)

Ratfor in the *FORTRAN Programmer's Guide*

NAME

rcp – remote file copy

SYNOPSIS

rcp *filename1 filename2*

rcp [**–r**] *filename ... directory*

DESCRIPTION

rcp copies files between machines. Each *filename* 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 a *filename* 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 by *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 copy files relative to the home directory of the user named *rname*, rather than the current user name on the remote host.

OPTIONS

–p Preserve modification times and access times.

–r copy each subtree rooted at *filename*; in this case the destination must be a directory.

SEE ALSO

ftp(1C), rsh(1C), rlogin(1C)

BUGS

rcp is meant to copy between different hosts; attempting to *rcp* a file onto itself (as with "myhost% *rcp tmp/file myhost:/tmp/file*") results in a severely corrupted file.

rcp doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

rcp can become confused by output generated by commands in a *.profile*, *.cshrc*, or *.login* 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.

If you forget to quote metacharacters intended for the remote host you get an incomprehesible error message.

NAME

rdist – remote file distribution program

SYNOPSIS

rdist [–nqbRhivwy] [–f distfile] [–d var=value] [–m host] [name ...]

rdist [–nqbRhivwy] -c name ... [login@]host[:dest]

DESCRIPTION

Rdist is a program to maintain identical copies of files over multiple hosts. It preserves the owner, group, mode, and mtime of files if possible and can update programs that are executing. *Rdist* reads commands from *distfile* to direct the updating of files and/or directories. If *distfile* is '–', the standard input is used. If no **–f** option is present, the program looks first for 'distfile', then 'Distfile' to use as the input. If no names are specified on the command line, *rdist* will update all of the files and directories listed in *distfile*. Otherwise, the argument is taken to be the name of a file to be updated or the label of a command to execute. If label and file names conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

The **–c** option forces *rdist* to interpret the remaining arguments as a small *distfile*. The equivalent distfile is as follows.

```
( name ... ) -> [login@]host
        install   [dest] ;
```

Other options:

–d Define *var* to have *value*. The **–d** option is used to define or override variable definitions in the *distfile*. *Value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and/or spaces.

–m Limit which machines are to be updated. Multiple **-m** arguments can be given to limit updates to a subset of the hosts listed in the *distfile*.

–n Print the commands without executing them. This option is useful for debugging *distfile*.

–q Quiet mode. Files that are being modified are normally printed on standard output. The **–q** option suppresses this.

–R Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truely identical copies of directories.

–h Follow symbolic links. Copy the file that the link points to rather than the link itself.

–i Ignore unresolved links. *Rdist* will normally try to maintain the link structure of files being transfered and warn the user if all the links cannot be found.

–v Verify that the files are up to date on all the hosts. Any files that are out of date will be displayed but no files will be changed nor any mail sent.

–w Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This will preserve the directory structure of the files being copied instead of flattening the directory structure. For example, renaming a list of files such as (dir1/f1 dir2/f2) to dir3 would create files dir3/dir1/f1 and dir3/dir2/f2 instead of dir3/f1 and dir3/f2.

–y Younger mode. Files are normally updated if their *mtime* and *size* (see *stat*(2)) disagree. The **–y** option causes *rdist* not to update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

–b Binary comparison. Perform a binary comparison and update files if they differ rather than

comparing dates and sizes.

Distfile contains a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

<variable name> '=' <name list>
[label:] <source list> '->' <destination list> <command list>
[label:] <source list> '::' <time_stamp file> <command list>

The first format is used for defining variables. The second format is used for distributing files to other hosts. The third format is used for making lists of files that have been changed since some given date. The *source list* specifies a list of files and/or directories on the local host which are to be used as the master copy for distribution. The *destination list* is the list of hosts to which these files are to be copied. Each file in the source list is added to a list of changes if the file is out of date on the host being updated (second format) or the file is newer than the time stamp file (third format).

Labels are optional. They are used to identify a command for partial updates.

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with '#' and end with a newline.

Variables to be expanded begin with '$' followed by one character or a name enclosed in curly braces (see the examples at the end).

The source and destination lists have the following format:

<name>

or

'(' <zero or more names separated by white-space> ')'

The shell meta-characters '[', ']', '{', '}', '*', and '?' are recognized and expanded (on the local host only) in the same way as *csh*(1). They can be escaped with a backslash. The '~' character is also expanded in the same way as *csh* but is expanded separately on the local and destination hosts. When the –w option is used with a file name that begins with '~', everything except the home directory is appended to the destination name. File names which do not begin with '/' or '~' use the destination user's home directory as the root directory for the rest of the file name.

The command list consists of zero or more commands of the following format.

'install' <options> opt_dest_name ';'
'notify' <name list>';'
'except' <name list>';'
'except_pat' <pattern list>';'
'special' <name list>string ';'

The *install* command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. *Opt_dest_name* is an optional parameter to rename files. If no *install* command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name will be created if they do not exist on the remote host. To help prevent disasters, a non-empty directory on a target host will never be replaced with a regular file or a symbolic link. However, under the '–R' option a non-empty directory will be removed if the corresponding filename is completely absent on the master host. The *options* are '–R', '–h', '–i', '–v', '–w', '–y', and '–b' and have the same semantics as options on the command line except they only apply to the files in the source list. The login name used on the destination host is the same as the local host unless the destination name is of the format "login@host".

The *notify* command is used to mail the list of files updated (and any errors that may have occured) to the listed names. If no '@' appears in the name, the destination host is appended to the name (e.g., name1@host, name2@host, ...).

The *except* command is used to update all of the files in the source list **except** for the files listed in *name list*. This is usually used to copy everything in a directory except certain files.

The *except_pat* command is like the *except* command except that *pattern list* is a list of regular expressions (see *ed*(1) for details). If one of the patterns matches some string within a file name, that file will be ignored. Note that since '\' is a quote character, it must be doubled to become part of the regular expression. Variables are expanded in *pattern list* but not shell file pattern matching characters. To include a '$', it must be escaped with '\'.

The *special* command is used to specify *sh*(1) commands that are to be executed on the remote host after the file in *name list* is updated or installed. If the *name list* is omitted then the shell commands will be executed for every file updated or installed. The shell variable 'FILE' is set to the current filename before executing the commands in *string*. *String* starts and ends with '"' and can cross multiple lines in *distfile*. Multiple commands to the shell should be separated by ';'. Commands are executed in the user's home directory on the host being updated. The *special* command can be used to rebuild private databases, etc. after a program has been updated.

The following is a small example.

```
HOSTS = ( matisse root@arpa )

FILES = ( /bin /lib /usr/bin /usr/games
        /usr/include/{*.h,{stand,sys,vax*,pascal,machine}/*.h}
        /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )

EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
        sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )

${FILES} -> ${HOSTS}
        install -R ;
        except /usr/lib/${EXLIB} ;
        except /usr/games/lib ;
        special /usr/lib/sendmail "/usr/lib/sendmail -bz" ;

srcs:
/usr/src/bin -> arpa
        except_pat ( \\.o\$ /SCCS\$ ) ;

IMAGEN = (ips dviimp catdvi)

imagen:
/usr/local/${IMAGEN} -> arpa
        install /usr/local/lib ;
        notify ralph ;

${FILES} :: stamp.cory
        notify root@cory ;
```

FILES

distfile	input command file
/tmp/rdist*	temporary file for update lists

SEE ALSO

sh(1), csh(1), stat(2)

DIAGNOSTICS

A complaint about mismatch of rdist version numbers may really stem from some problem with starting your shell, e.g., you are in too many groups.

BUGS

Source files must reside on the local host where rdist is executed.

There is no easy way to have a special command executed after all files in a directory have been updated.

Variable expansion only works for name lists; there should be a general macro facility.

Rdist aborts on files which have a negative mtime (before Jan 1, 1970).

There should be a 'force' option to allow replacement of non-empty directories by regular files or symlinks. A means of updating file modes and owners of otherwise identical files is also needed.

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

SEE ALSO

addbib(1), indxbib(1), lookbib(1), roffbib(1), sortbib(1)

NAME

screenblank – turn off video when the mouse and keyboard are idle

SYNOPSIS

screenblank [**–m**] [**–k**] [**–d** *interval*] [**–e** *interval*] [**–f** *frame_buffer*]

DESCRIPTION

screenblank turns off the display when the mouse and keyboard are idle for an extended period (the default is 10 minutes).

OPTIONS

–m Do not check whether the mouse has been idle.

–k Do not check whether the keyboard has been idle.

–d *interval*

Disable after *interval* seconds. *interval* is a number of the form *xxx.xxx* where each *x* is a decimal digit. The default is 600 seconds (10 minutes).

–e *interval*

Enable within *interval* seconds. *interval* is the time between successive polls for keyboard or mouse activity. If a poll detects keyboard or mouse activity, the display is resumed. *interval* is a number of seconds, of the form *xxx.xxx* where each *x* is a decimal digit. The default is 0.25 seconds.

–f *frame_buffer*

frame_buffer is the path name of the frame buffer on which video disabling/enabling applies. The defaults is /dev/fb.

SEE ALSO

lockscreen(1)

BUGS

When not running *suntools*(1), only the RETURN key resumes video display.

NAME

screendump – dump frame buffer image to file

SYNOPSIS

screendump [**–ce**] [**–f** *framebuffer*] [**–t** *type*] [*file*]

DESCRIPTION

Screendump reads the contents of a frame buffer and writes the display image to *file* (standard output default) in Sun standard rasterfile format (see */usr/include/rasterfile.h*).

OPTIONS

–c Dump the frame buffer contents directly without making a temporary copy in a memory pixrect. Saves time and memory but lengthens the time the frame buffer must be inactive to guarantee a consistent screen dump.

–f *framebuffer*
Dump the specified frame buffer device (default /dev/fb).

–t *type* Set the output rasterfile type (default 1, RT_STANDARD). See */usr/include/rasterfile.h*.

–e Set the output rasterfile type to 2, RT_BYTE_ENCODED. For most images this saves a significant amount of space compared to the standard format.

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/cgtwo0 save.color.image**

writes the current contents of the color frame buffer */dev/cgtwo0* 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

/usr/lib/rasfilters/* Filters for non-standard rasterfile formats

SEE ALSO

lpr(1), rasfilter8to1(1), rastrepl(1), screenload(1)

File I/O Facilities for Pixrects in *Pixrect Reference Manual*

BUGS

The output file or the screen may be corrupted if the frame buffer contents are modified while the dump is in progress.

NAME

screenload – load frame buffer image from file

SYNOPSIS

screenload [**–dp**] [**–f** *framebuffer*] [**–bgw**] [**–h** *count data ...*] [**–i** *color*] [*file*]

DESCRIPTION

Screenload reads the Sun standard rasterfile *file* (see */usr/include/rasterfile.h*) and displays its contents on a frame buffer. *Screenload* is able to display monochrome images on a color display, but cannot display color images on a monochrome display. If the input file contains a color image, a frame buffer has not been explicitly specified, and /dev/fb is a monochrome frame buffer, *screenload* will look for a color frame buffer with one of the standard names.

If the image contained in the input file is larger than the actual resolution of the display, *screenload* clips the right and bottom edges of the input image. If the input image is smaller than the display (for example, loading an 1152-by-900 image on a 1600-by-1280 high resolution 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.

OPTIONS

–d Print a warning message if the display size does not match the rasterfile image.

–p Wait for a newline to be typed on the standard input before exiting.

–f *framebuffer*
Display the image on the specified frame buffer device (default /dev/fb).

–b Fill the border area 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 Fill the border area with a pattern of "desktop grey". On a monochrome display this results in a border matching the default background pattern used by SunView; 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.

–w Fill the border area 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 *count data ...*
Fill the border area with the bit pattern described by the following *count* 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.

–i *color* Fill the border area with the given color value (default 255).

EXAMPLES

tutorial% **screenload saved.display.image**

loads 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/cgtwo0 monochrome.image**

reloads the raster image in the file *monochrome.image* on the color frame buffer device */dev/cgtwo0*.

tutorial% **screenload –h1 ffff small.saved.image**

is equivalent to the **–b** option (fill border with black), while

tutorial% **screenload –h4 8888 8888 2222 2222 small.saved.image**

is equivalent to the **–g** option (fill border with desktop grey).

FILES

/usr/lib/rasfilters/* Filters for non-standard rasterfile formats

SEE ALSO

rasfilter8to1(1), rastrepl(1), screendump(1), screenload(1)

File I/O Facilities for Pixrects in *Pixrect Reference Manual*

NAME

sh – shell, the standard UNIX command interpreter and command-level language

SYNOPSIS

sh [**–acefhiknstuvx**] [*arguments*]

DESCRIPTION

sh, the Bourne shell, is the standard UNIX command interpreter. It executes commands read from a terminal or a file.

Definitions

A *blank* is a TAB or a SPACE character. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, –, $, and ! .

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is –, commands are initially read from *$HOME/.profile*, if such a file exists and is owned by you. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*.

OPTIONS

The flags below are interpreted by the shell on invocation only; unless the **–c** or **–s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters for use with the commands that file contains.

–c *string* If the **–c** flag is present commands are read from *string*.

–s If the **–s** flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) 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, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command, under *Special Commands*, below.

USAGE

Refer to *Doing More With UNIX Beginner's Guide* for more information about using the shell as a programming language.

Commands

A *simple command* is a sequence of nonblank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a command is its exit status if it terminates normally, or (octal) 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 | (or, for historical compatibility, 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 normally waits for the last command to terminate before prompting for or accepting the next input line. The exit status of a pipeline is the exit status of its last command.

A *list* is a sequence of one or more simple commands or pipelines, separated by ;, &, &&, or | | , and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and | | . The symbols && and | | also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (the shell does *not* wait for that pipeline to finish). The symbols && and | | are used to indicate condition execution of the list that follows. With && , *list* is executed only if the preceding pipeline (or command) returns a zero exit status. With | | , *list* is executed only if the preceding pipeline (or

command) returns a nonzero exit status. An arbitrary number of NEWLINEs may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple command or one of the following constructions. Unless otherwise stated, the value returned by a command is that of the last simple command executed in the construction.

for *name* [**in** *word* ...] **do** *list* **done**
Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see *Filename Generation*) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a subshell.

{*list*;} *list* is simply executed.

name () {*list*;}
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a NEWLINE to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing NEWLINE s are removed.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
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 * or @, all the positional parameters, starting with $1, are substituted (separated by spaces). Parameter $0 is set from argument zero when the shell is invoked.

If the colon (:) is omitted from the following expressions, the shell only checks whether *parameter* is set or not.

${*parameter*:-*word*}

If *parameter* is set and is nonnull, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}

If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}

If *parameter* is set and is nonnull, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}

If *parameter* is set and is nonnull, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last synchronously executed command.

$ The process number of this shell.

! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below).

CDPATH
The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename can be followed by **%** and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "$ ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally SPACE, TAB, and NEWLINE.

SHELL When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the filename part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ``) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The

character . at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > NEWLINE SPACE TAB

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \NEWLINE is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, `, ", and $. "$*" is equivalent to "$1 $2 ...", whereas "$@" is equivalent to "$1" "$2" ....

Prompting

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a NEWLINE is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of PS2) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.
>>*word*	Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<[–]*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) \NEWLINE is ignored, and \ must be used to quote the characters \, $, `, and the first character of *word*. If – is appended to <<, all leading TABs are stripped from *word*, and from the document.
<&*digit*	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using >&*digit*.
<&–	The standard input is closed. Similarly for the standard output using >&–.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by & the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see *environ*(5V)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd
```

and

```
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set –k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying */bin*, and */usr/bin*, in addition to the current directory). Directories are searched in order. The the current directory is specified by a null path name, which can appear immediately after the equal sign (**PATH=:...**) or between the colon delimiters (...::...) anywhere else in the path list. If the command name contains a / the search path is not used; such commands will not be executed by a restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an binary executable (see *a.out*(5) for details) it is assumed to be a file containing shell commands. A subshell is spawned to read it. A parenthesized command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified 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*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*.

echo [*arg* ...]
Echo arguments. See *echo*(1V) for usage and description.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

hash [**−r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

login [*arg* ...]
Equivalent to **exec login** *arg* .... See *login*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**–aefhkntuvx–** [*arg* ...]]
–a Mark variables which are modified or created for export.
–e Exit immediately if a command exits with a nonzero exit status.
–f Disable filename generation
–h Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
–k All keyword arguments are placed in the environment for a command, not just those that precede the command name.
–n Read commands but do not execute them.
–t Exit after reading and executing one command.
–u Treat unset variables as an error when substituting.
–v Print shell input lines as they are read.
–x Print commands and their arguments as they are executed.
– Do not change any of the flags; useful in setting **$1** to –.
Using + rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, and so on. If no arguments are given, the values of all names are printed.

shift [*n*]
The positional parameters are shifted to the left, from position *n*+1 to position 1, and so on. Previous values between **$1** and **$***n* are discarded. If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test*(1V) for usage and description.

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent 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 the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

umask [*ooo*]
The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively. The value of each specified digit is subtracted from the corresponding 'digit' specified by the system for the creation of a file. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**). The current value of the mask is printed if *ooo* is omitted.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS** cannot be unset.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

$HOME/.profile
*/tmp/sh**
/dev/null

SEE ALSO

csh(1), cd(1), echo(1V), login(1), pwd(1), test(1V), dup(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(5), profile(5), environ(5).

CAVEATS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

NAME

suntools, othertools, selection_svc – the SunView window environment

SYNOPSIS

suntools [**–n** | **–s** *startup-file*] [**–S**] [**–d** *display-device*] [**–m** *mouse-device*] [**–k** *keyboard-device*] [**–p**] [**–b** *red green blue*] [**–f** *red green blue*] [**–i**] [**–B** | **–F** | **–P**] [**–pattern on** | **off** | **gray** | *iconedit-file-name*] [**–background** *raster-file-name*] [**–8bit_color_only**] [**–toggle_enable**] [**–overlay_only**]

GETTING STARTED

suntools starts up the SunView environment and (unless you have specified otherwise) a default arrangement of a few useful "tools," or window-based utilities.

See *Start-up Processing* below to learn how to specify your own initial arrangement of tools. Some of the behavior of *suntools* is controlled by settings in your defaults database; see *SunView Defaults* below.

OPTIONS

–n Bypass startup processing by ignoring both the */usr/lib/suntools* and *~/.suntools* files. See *Startup Processing* for details.

–s *startup-file*
Read startup commands from *startup file* (instead of */usr/lib/suntools* or *~/.suntools*).

–S Set "click-to-type" mode, allowing you to select a window by clicking in it. Having done so, input is directed to that window regardless of the position of the mouse-cursor, until you click to select some other window.

–d *display-device*
Use *display device* as the output device on which to run, rather than the default frame buffer device, */dev/fb*.

–m *mouse-device*
Use *mouse device* as the system pointing device (locator), rather than the default mouse device, */dev/mouse*.

–k *keyboard-device*
Accept keyboard input from *keyboard device*, rather than the default keyboard device, */dev/kbd*.

–p Prints to standard out the name of the window device used for the *suntools* Root Window.

–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). Prism users that use this option should use the **-8bit_color_only** option too.

–f *red green blue*
Specifies the values of the *red, green* and *blue* components of the foreground color. If this option is not specified, each component of the foreground color is 0 (black). Prism users that use this option should use the **-8bit_color_only** option too.

–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.

–B Use the background color for the Root Window color.

–F Use the foreground color for the Root Window color.

–P Use a stipple pattern for the Root Window color. This option is assumed unless **–F** or **–B** is specified.

–pattern [on | off | gray | *iconedit-file-name*]
Use the indicated "pattern" to cover the Root Window. **on** means to use the default desktop gray pattern. **off** means to not use the default desktop gray pattern. **gray** means to use a 50% gray

color on color monitors. *iconedit-file-name* is the name of a file produced with *iconedit*(1) which contains an image that is replicated all over the Root Window.

–background *raster-file-name*
Use the indicated raster file as the image in your Root Window. The raster file can be created with *screendump*(1). Screen dumps produced on color monitors currently do not work as input to this option. Small images are centered on the screen.

–8bit_color_only
For multiple plane group frame buffers, only let windows be created in the 8 bit color plane group. This frees up the black and white overlay plane to have a separate desktop running on it. This option is usually used with the **–toggle_enable** option. See the section below entitled *Multiple Desktops on the Same Screen.*

–toggle_enable
For multiple plane group frame buffers, when sliding the cursor between different desktops running within different plane groups on the same screen, change the enable plane to allow viewing of the destination desktop. See the section below entitled *Multiple Desktops on the Same Screen.*

–overlay_only
For multiple plane group frame buffers, only let windows be created in the black and white overlay plane group. This frees up the 8 bit color plane group to have a separate desktop running in it. This option is usually used with the **–toggle_enable** option. See the section below entitled *Multiple Desktops on the Same Screen.*

DESCRIPTION

Windows

The *SunView* environment always has one window open, called the *Root Window*, which covers the whole screen. A solid color is its only content. Each tool is given its own window which lies on top of some of the Root Window (and possibly on top of other tools). A window obscures any part of another window which lies below it.

Input to Windows

Mouse input is always directed to the window the mouse cursor is in. You can have keyboard input follow mouse input, or you can use the "Click-to-Type" approach. With Click-to-Type, keyboard input continues to be directed to a window, no matter where the mouse is pointing, until you click the left or middle mouse button in another window. Click-to-Type is an option in your defaults database; see *SunView Defaults* below. If you are not using Click-to-Type, and your mouse cursor is in the Root Window, keyboard input is discarded.

Your input actions (mouse motions, button pushes, and keystrokes) are synchronized. This means that you can "type-ahead" and "mouse-ahead," even across windows.

The Mouse Buttons

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.

Menus

suntools provides *pop-up menus*. In the current release, there are two styles of pop-up menus: the original menu style, called *stacking menus*, and a new style, called *walking menus* (also known as "pull-right menus"). A menu is invoked by pressing and holding the menu button. The menu remains on the screen as long as you hold the menu button down. To select a menu item, point at it (it will be highlighted), then release the menu button.

With stacking menus, more than one menu can appear simultaneously. The menus are shown in a stack, with the label of each menu visible, and with the current menu on top so that its items are visible. To bring a menu to the top (and make its items available), select its label as you would a menu item. Then push the menu button again. The menu stack is repainted with the selected menu on top.

With walking menus, any menu item can have an arrow (=>) on the right. Pointing to this arrow invokes a *sub-menu*, with additional menu items that can be selected. Selecting an item that has an arrow (a "pull-right item") invokes the first item on the sub-menu.

Walking menus are an option in your defaults database; see *SunView Defaults* below.

The Root Window Menu

You can use the default Root Window Menu to start ten common tools and perform three functions. To invoke it, hold down the menu button when the mouse cursor is anywhere in the Root Window.

The items in the default Root Window Menu are:

ShellTool Creates a new *shelltool*(1), running a new copy of the shell.

CommandTool Creates a new *cmdtool*(1), a scrollable cousin of the *shelltool*.

MailTool Creates a new *mailtool*(1).

TextEditor Creates a new *textedit*(1).

DefaultsEditor Creates a new *defaultsedit*(1), for browsing or changing your defaults database.

IconEditor Creates a new *iconedit*(1).

DbxTool Creates a new *dbxtool*(1), a window-based debugger.

PerfMeter Creates a new *perfmeter*(1), to monitor system performance.

GraphicsTool Creates a new *gfxtool*(1), for running graphics programs.

Console Creates a new Console window, a *cmdtool* with a –C flag, which acts as the system console. In particular, most error messages will be directed to the console. You should always have a console window on your screen.

Lock Screen Completely covers the screen with a graphics display, and "locks" the workstation until you type your password. When you "unlock" the workstation, the screen is restored as it was when you locked it. See *lockscreen*(1) for details.

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 SunView system.

Exit Suntools Exits the *suntools* program. Closes all tool windows and kills their associated processes (depending on the processes, this can be fairly slow). 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 Suntools command; press the right button to cancel.

You can specify your own Root Window Menu; see *SunView Defaults* below.

The Frame Menu

A small set of universal functions are available through the Frame Menu. There are also accelerators for some of these functions; see below.

You can invoke the Frame Menu when the cursor is over any part of the tool which does not provide an application-specific menu, such as the tool namestripe (black stripe holding the tool's name), the border stripes of the window, and the whole of the tool's icon.

The items in the Frame Menu are:

Close (Open) Only one of **Close** or **Open** appears in the menu, depending on the current state of the window. **Close** shrinks the tool to a small image (an *icon*). **Open** reopens an icon and places the tool in the spot it occupied when it was open. Icons are placed on the screen according to the icon policy in your defaults database; see *SunView Defaults* below. You can move a closed window just like an open window. When the window is closed, the tool's process(es) continue to run.

Move Moves the tool window to another spot on the screen. When invoked, **Move** instructs

you with an instruction box that appears in the middle of the screen.
If you are using walking menus, **Move** has a sub-menu with two items: **Constrained** and **Unconstrained. Constrained** moves are either vertical or horizontal, but not both. Selecting **Move** invokes a **Constrained** move.

Resize Shrinks or stretches the size of a window on the screen. **Resize**, like **Move**, instructs you with an instruction box that appears in the middle of the screen.
If you are using walking menus, **Resize** has a sub-menu with four items: **Constrained, Unconstrained, Zoom** (or **UnZoom**, depending on the current state of the window) and **FullScreen. Constrained** resizes are either vertical or horizontal, but not both. **Zoom** makes a window the full height of the screen; **UnZoom** undoes this. **FullScreen** makes a window the full height and width of the screen; **UnZoom** undoes this. Selecting **Resize** invokes a **Constrained** resize.

Expose Brings the window to 'the top of the pile'. The whole window becomes visible, and occludes any window it happens to overlap on the screen.

Hide Puts the window on the 'bottom of the pile'. The window is occluded by any window which overlaps it.

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 Suntools** command in the Root Window Menu.

Frame Menu Accelerators

Accelerators are provided for some of the Frame Menu functions. You can invoke these functions quickly with a simple button push in the tool window's name stripe or outer boundary, without displaying a menu. See *Windows and Window-Based Tools: Beginner's Guide* for more details.

The accelerators for the various functions are:

Open Click the select mouse button when the cursor is over the icon.

Move Depress the adjust mouse button while the cursor is in the tool's name stripe or outer boundary. A bounding box is displayed which tracks the mouse as long as you hold the adjust button down.
If the cursor is near a corner when you press the mouse button, the move is **Unconstrained.** If it is in the middle third of a side, the move is **Constrained.**

Resize While holding down the CTRL key, depress the adjust mouse button while the cursor is in the tool's name stripe or outer boundary. A bounding box is displayed, one side or corner of which tracks the mouse as long as you hold the adjust button down.
If the cursor is near a corner when you press the mouse button, the resize is **Unconstrained.** If it is in the middle third of a side, the resize is **Constrained.**

Zoom (UnZoom) While holding down the CTRL key, click the select mouse button while the cursor is in the tool's name stripe or outer boundary.

Expose Click the select mouse button while the cursor is on the tool's name stripe or outer boundary.

Hide While holding down the SHIFT key, click the select mouse button while the cursor is on the tool's name stripe or outer boundary.

In addition, you can use two function keys as even faster accelerators. To expose a window that is partially hidden, hit the Expose key (normally L5) while the cursor is anywhere in the tool window, *not just* on the tool's name stripe or outer boundary. Or, if the window is completely exposed, use the Expose key to hide it. Similarly, to close an open window, hit the Open key (normally L7) while the cursor is anywhere in the tool window, *not just* on the tool's name stripe or outer boundary. Or, if the window is iconic, use the Open key to open it. You can change which keys mean Expose and Open by using *setkeys*(1).

In many multi-subwindow tools, you can adjust the boundary between two subwindows up or down without changing the overall size of the tool. While holding down the CTRL key, depress the adjust mouse button over the boundary. A bounding box is displayed for the subwindow selected. Adjust the size of that subwindow, exactly as with the **Resize** operation.

Startup Processing: The .suntools File

Unless you override it, *suntools* will start up with a predefined arrangement of windows. The default arrangement is specified by the file */usr/lib/suntools*. If there is a file called *.suntools* in your home directory, that will be used instead. The **–s** flag on the command line indicates that the initial window arrangement should be read from a file with a different name. The **–n** switch suppresses this start-up processing altogether.

To create your own *.suntools*, arrange the screen the way you like, then save the arrangement by running *toolplaces* and redirecting its standard output to *.suntools*. See *toolplaces*(1) for a description of the format of this file, or take a look at */usr/lib/suntools*.

SunView Defaults

SunView allows you to customize the behavior of tools and packages by setting options in a defaults database (one for each user). Use *defaultsedit*(1) to browse and edit your defaults database. Select the "SunView" category to see the following items:

Walking_menus — If enabled, the Root Window Menu, the Frame Manager Menu, and many tools will use walking menus. Tools that have not been converted will still use stacking menus. If disabled, all tools will use stacking menus. Default value is "Disabled".

Click_to_Type — If enabled, keyboard input will stay in a window until you click the left or middle mouse button in another window. If disabled, keyboard input will follow the mouse. Default value is "Disabled".

Font — You can change the SunView default font by giving the full pathname of the font you want to use. Some alternate fonts are in the directory */usr/lib/fonts/fixedwidthfonts*. The previous (2.0 release) default font is */usr/lib/fonts/fixedwidthfonts/screen.r.13*. Default value is null, which gives you the same effect as if you had specified */usr/lib/fonts/fixedwidthfonts/screen.r.11*.

Rootmenu_filename — You can change the Root Window Menu by giving the full pathname of a file that specifies your own menu. See *Customizing the Root Window Menu* below for details. Default value is null, which gives you the menu found in */usr/lib/rootmenu*.

Icon_gravity — Determines which edge of the screen ("North", "South", "East", or "West") icons will place themselves against. Default value is "North".

Icon_close_level — Determines whether icons will close ahead of or behind other windows and icons. Default value is "Ahead_of_all".

Jump_cursor_on_resize — If enabled, during a resize the cursor will jump to the edge of the window. If disabled, the window edge will move to the current location of the cursor. Default value is "Disabled".

Audible_bell — If enabled, the "bell" command will result in a beep. Default value is "Enabled".

Visible_bell — If enabled, the "bell" command will cause the screen to flash. Default value is "Enabled".

Embolden_Labels — If enabled, all tool labels are boldface. Default value is "Disabled".

Root_Pattern — Used to specify the "pattern" that covers the Root Window. "on" means to use the default desktop gray pattern. "off" means to not use the default desktop gray pattern. "gray" means to use a 50% gray color on color monitors. Anything else is the name of

a file produced with *iconedit*(1) which contains an image that is replicated all over the Root Window. Default value is "on".

After you have set the options you want, click on the Save button in *defaultsedit*; then exit *suntools* and restart it.

Customizing the Root Window Menu

The file called */usr/lib/rootmenu* contains the specification of the default Root Window Menu. You can change the Root Window Menu by creating your own file and giving its name in the Rootmenu_filename item in the *SunView Defaults* (see above).

Lines in the file have the following format: The left side is a menu item to be displayed; the right side is a command to be executed when that menu item is invoked. You can also include comment lines (beginning with a '#') and blank lines.

If you are using stacking menus ("Walking_menus Disabled" in SunView defaults), the menu item must be a string (strings with embedded blanks must be delimited by double quotes). If you are using walking menus ("Walking_menus Enabled"), the menu item can be a string or the full pathname of an icon file, delimited by angle brackets. With care, strings and icons can be mixed in one menu.

There are four reserved-word commands that can appear on the right side.

EXIT — Exit the *suntools* program, after user confirmation.

REFRESH — Redraw the entire screen.

MENU — If you are using stacking menus, a menu is added to the pile with the Root Window Menu. The menu contents are taken from the filename that follows the MENU command. You must give the full pathname of the file.
If you are using walking menus, this menu item is a pull-right item with a submenu. If a filename follows the MENU command, the submenu contents are taken from the filename. Otherwise, all the lines between this MENU command and a matching END command are added to the submenu.

END — Marks the end of a nested submenu. The left side of this line should match the left side of a line with a MENU command. Not valid if you are using stacking menus.

If the command is not one of these four reserved-word commands, it is treated as a command line and executed. No shell interpretation is done, although you can run a shell as a command.

Here is a menu file that demonstrates some of these features:

```
Quit                    EXIT
Clock                   clock -r -f
"Mail reader"           mailtool
"More tools"            MENU  /usr/foo/me/moretools.menu
"Click to type"         swin -c
"Follow mouse"          swin -m
"Print selection"       sh -c get_selection | lpr
# Only if you are using walking menus:
"Nested menu"           MENU
    Cmdtool             cmdtool
    Shelltool           shelltool
"Nested menu"           END
"Icon menu"             MENU
```

```
        </usr/include/images/textedit.icon>        textedit
        </usr/include/images/iconedit.icon>        iconedit
    "Icon menu"          END
```

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 -n &". This starts *suntools* on a 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, and back when you move the cursor off the left hand side of the color screen.

Multiple Desktops on the Same Screen

Given appropriate hardware, the *suntools* program can be made to run separate *desktops* on the same screen. This facility is an extension of the features described in the previous section entitled *Multiple / Color Displays*. The Prism is an example of a machine with *multiple plane groups* that can take advantage of this facility. Each plane group on a machine may have its own invocation of *suntools* running on it. Such an invocation is called a desktop. The keyboard and mouse input devices are shared among multiple desktops. The mouse cursor slides from one desktop to another when you move the cursor off the edge of the screen.

A common multiple desktop configuration for the Prism is one monochrome and one color desktop. You could set up an instance of *suntools* on each plane group in the following way:

1. Invoke suntools in the color plane group by running "suntools -8bit_color_only -toggle_enable". This starts *suntools* on the default frame buffer named */dev/fb* but limits access to the color plane group.
2. In a *shelltool*, run "suntools -d /dev/bwtwo0 -toggle_enable -n &". This starts *suntools* in the overlay plane that is accessed by /dev/bwtwo0.
3. In a *shelltool* run "adjacentscreens -c /dev/fb -l /dev/bwtwo0". This sets up cursor tracking so that the cursor slides from the monochrome desktop to the color desktop when you move the cursor off the right hand side of the monochrome desktop, and back when you move the cursor off the left hand side of the color desktop.

Old pre-3.2 applications run on the 8bit_color_only desktop will not appear because they will be writing to the overlay plane. I.e., don't run old pre-3.2 applications on an 8bit_color_only desktop.

There is an application called the *switcher* that is used as an alternative to *adjacentscreens* for getting between desktops on the Prism. Clicking the *switcher* icon gets you to another desktop using some amusing video wipe type animation. The *switcher* can also be used to simply set the enable plane to 0 or 1 if the enable plane get out of wack. See the man page *switcher*|(1) for details.

Generic Tool Arguments

Most window-based applications 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 (closed)
-Wt	(-font)	filename	
-Wn	(-no_name_stripe)		
-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.

Getting Out

To exit any tool, invoke the **Quit** command in the Frame Menu as described above. To exit the entire window system, invoke **Exit Suntools** in the Root Window 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 if you inadvertently start *suntools* without a mouse attached to the system.

SEE ALSO

Windows and Window-Based Tools: Beginner's Guide

Some of the applications that run in the SunView environment:

clock(1), *cmdtool*(1), *dbxtool*(1), *defaultsedit*(1), *fontedit*(1), *gfxtool*(1), *iconedit*(1), *lockscreen*(1), *mailtool*(1), *overview*(1), *perfmeter*(1), *perfmon*(1), *shelltool*(1), *tektool*(1), *textedit*(1), *traffic*(1)

Some of the utility programs that run in or with the SunView environment:

adjacentscreens(1), *clear_functions*(1), *get_selection*(1), *rastps*(1), *setkeys*(1), *stty_from_defaults*(1), *swin*(1), *switcher*(1), *toolplaces*(1)

ENVIRONMENT

DEFAULTS_FILE

The value of this environment variable indicates the file from which SunView defaults are read. When it is undefined, defaults are read from the *.defaults* file in your home directory.

FILES

~/.suntools
/usr/bin/suntools
/usr/bin/othertools
/usr/bin/get_selection
/usr/bin/selection_svc
/usr/lib/suntools
/usr/lib/rootmenu
/usr/lib/fonts/fixedwidthfonts/*
/dev/win*x*
/dev/pty*px*
/dev/tty*px*

/dev/fb
/dev/kbd
/dev/mouse
/etc/utmp

BUGS

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 Window Menu. Then invoke the **Console** item to start a console.

To improve interactive performance, the kernel should be reconfigured in order to make more memory available for applications. See the *System Manager's Guide*.

With an optical mouse, sometimes the arrow-shaped cursor will not move at start-up; moving the mouse in large circles on its special pad for a few seconds will bring the cursor to life.

suntools 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.

On a color display, all of the colors may "go strange" when the cursor is in certain windows. This is caused by SunView accommodating a particular window's request for a large number of colors.

When running multiple desktops, be careful to not have more than one *shelltool* or *cmdtool* acting as the console at once. Kill one console before starting another.

NAME

swin – set/get SunView user input options

SYNOPSIS

swin [**–c**] [**–g**] [**–h**] [**–m**] [**-r** *event value shift_state*] [**-s** *event value shift_state*] [**-t** *seconds*]

DESCRIPTION

The *swin* (set window; analogous to *stty(1)*) command lets you change some of the input behavior of your SunView environment. By default, your keyboard input follows your mouse cursor. This means that in order to type to a window you position the mouse cursor over the window. This is called *keyboard-follows-mouse* mode.

You can specify that the keyboard input continues to go to the same window, regardless of the mouse cursor position, until you take some specific action, like clicking the mouse. When this is done, you can roam around the screen with the mouse cursor and not change the window to which keyboard input is directed. Running SunView like this is said to be operating in *click-to-type* mode.

When running in click-to-type mode, one user action *sets* the type-in point in the window that you want to receive keyboard input. The default user action to do this is the pressing of the left mouse button while positioning the mouse cursor over the new type-in point. This user action can be changed.

Another user action *restores* the previous type-in point in the window that you want to receive keyboard input. The default user action to do this is the pressing of the middle mouse button while positioning the mouse cursor over the window. This user action can be changed.

OPTIONS

–c Turn on click-to-type mode using the default user actions: the left mouse button sets the type-in point and the middle button restores the type-in point. You can use the *defaultsedit(1)* program to set click-to-type on permanently; see the SunView/Click_to_Type option.

–m Run in keyboard-follows-mouse mode.

–s *event value shift_state*

Set the user action that sets the type-in point and sets the keyboard input window. The *event* identifies the particular user action and is one of:

LOC_WINENTER
: the mouse cursor entering a window

MS_LEFT
: the left mouse button

MS_MIDDLE
: the middle mouse button

MS_RIGHT
: the right mouse button

decimal_number
: place the decimal number of a firm event here; see list of events in */usr/include/sundev/vuid_event.h* (avoid function keys, normally unused control-ascii characters are OK, normally unused shift keys are OK).

value identifies the transition of the *event* and is one of:

ENTER the mouse cursor entering a window (use with LOC_WINENTER)

DOWN the button associated with *event* went down

UP the button associated with *event* went up (avoid this)

The *shift_state* identifies the state of the shift keys at the time of the *event/value* pair in order for that pair to be used to control the keyboard input window. The *shift_state* is one of:

SHIFT_DONT_CARE

FILES

/usr/bin/switcher

SEE ALSO

suntools(1), shelltool(1), adjacentscreens(1)

NAME

symorder – rearrange name list

SYNOPSIS

symorder *orderlist symbolfile*

DESCRIPTION

orderlist is a file containing symbols to be found in symbolfile, 1 symbol per line.

symbolfile is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

This program was specifically designed to cut down on the overhead of getting symbols from /vmunix.

SEE ALSO

nlist(3)

NAME

graphics_demos, bouncedemo, cframedemo, framedemo, goban, jumpdemo, maze, shaded, show, showmap, spheresdemo, stringart, suncube – graphics demonstration programs

SYNOPSIS

bouncedemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

cframedemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

framedemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

goban *game*

jumpdemo [**–c**] [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

maze

shaded *object* [**–d** *dev*]

show *rasterfile* [*rasterfile* ...]

showmap [**–d** *dev*] [**–q**]

spheresdemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

stringart [**–d** *dev*] [**–q**]

suncube [**–d** *dev*] [**–q**]

DESCRIPTION

Note: Optional Software (Games and Demos Option). Refer to *Installing UNIX on the Sun Workstation* for information on how to install these demos.

Bouncedemo

bouncedemo displays a bouncing square.

Cframedemo

cframedemo displays a series of color frames, each of which contains a 256 by 256 image of eight-bit-deep pixels. *cframedemo* looks for the frames in the files *frame.1* through *frame.n* in the current working directory, and displays them in numerical order. When run in the directory */usr/demo/globeframes*, *cframedemo* displays a rotating view of the world.

Framedemo

framedemo displays a series of frames, each of which contains a 256 by 256 image one-bit-deep pixels (that is, the image is a square monochrome bitmap, with 256 bits on a side). *framedemo* looks for the frames in the files *frame.1* through *frame.n* in the current working directory, and displays them in numerical order. A set of sample frames is available in the directory */usr/demo/globeframes/**. *Interactive Commands*

If you move the cursor onto the image surface, you can type certain commands to affect the rate at which the frames are displayed. The initial rate is one frame per second:

f removes 1/20th of a second from the interval.

F removes one second from the interval. **Ff** makes the interval as small as possible.

s adds 1/20th of a second.

S adds one second.

Goban

goban is Japanese for "go board". It is an automatic board, but does not play go. If you invoke it with no *game* argument, *goban* displays an important historical game written about by the Nobel Prize winning author, Yasunari Kawabata in *The Master of Go*, a book which conveys the spirit of this ancient and facinating game.

Stones are placed on the board by selecting a grid point with the cursor and pressing the left-button. As stones are played, the color to play next alternates between black and white. The center-button, when pressed in the board area, backs up a move (undoes it). The right-button moves forward through the game's sequence of moves.

Stepping backward and forward does not alter the game until the left-button is pressed to place a stone, at which time a new branch in the line of play is begun. You can select branches by clicking the left button on moves with lettered labels on the board.

A text subwindow displays any commentary attached to a move. You can edit these comments, which are saved along with the game.

Jumpdemo

jumpdemo simulates the famous *Star Wars* jump to light-speed-sequence using vector drawing. Colored stars are drawn on color surfaces.

Maze

maze creates a random maze-pattern and tries a depth-first solution. If used in lockscreen, remember to run in "nice" mode since this demo consumes lots of cpu cycles.

Shaded

shaded displays shaded objects. Objects are located in */usr/demo/DATA* and include an icosahedron, glass, soccer ball, space shuttle, egg and pyramid. This demo can take up to 40 seconds to start up with som objects. Mouse input is required:

Interactive Commands

Click the left- and middle-buttons on the left grid to set the x-y orientation. Click the middle-button on the right grid to set the z orientation. Click the left-button away from either grid to open the features menu, from which you can make selections using the left-button.

After selecting the desired features, click the left-button away from all objects to exit the features menu.

Click the right-button to begin drawing the object. When the figure is finished, click the right-button to return to the grids and menu, or type **q** to exit.

Show

show displays rasterfiles in a window or on a raw screen. Sample files are contained in the directory */usr/demo/COLORPIX*. Running

show COLORPIX/*

from */usr/demo* will continuously cycle through the sample images.

Spheresdemo

spheresdemo computes a random collection of shaded spheres. Colored spheres are drawn on color surfaces.

Showmap

showmap displays 10 map projections continuously until interrupted. Each map is displayed for about 5 seconds. The maps are in the directory */usr/demo/MAPS*.

Stringart

stringart continuously displays a different "work of art" every 5 seconds. A total of 24336 different designs are possible. On color surfaces the designs will loop through the colors: red, olive, green, turquoise, blue, and violet.

Suncube

Displays a cube with the SUN logo mapped to each face. Will run continuously until interrupted. On color surfaces the colors of logo segments change gradually. On monochrome surfaces the logo segments remain hollow.

OPTIONS

–c Rotate the color map to produce a sparkling effect.

-d *surface*
Run the demo on a surface other than the window or system console, for instance: bouncedemo -d /dev/cgone0

-n*x* Draw *x* items, or repeat a sequence *x* times.

-r Retain the window. This allows the image to reappear when uncovered instead of restarting the demo.

-q Quick exit. Useful for running several demos from within a shell script.

SEE ALSO

gp_demos(6), gfxtool(1)

NAME

life – John Conway's game of life

SYNOPSIS

life

DESCRIPTION

Life is a program that plays John Conway's game of life. It only runs under *suntools*(1).

When invoked, *life* will display a window with a small control panel at the top, and a large drawing area at the bottom. You can create pieces in the drawing area with the left button, and erase them with the middle button. When you select **Run** in the control panel, the pieces will begin to evolve, and the drawing region will update itself at a speed controlled by the slider labeled with **Fast** and **Slow**. *Life* keeps track of all the pieces even if they are not visible. The scroll bars surrounding the drawing region can be used to see pieces that have moved out of view. There are some standard patterns that can be drawn by popping up a menu in the drawing subwindow.

The meaning of the items in the first row of the control panel (from left to right) are as follows. If you click on the picture which looks like a tic-tac-toe board, a grid will appear in the drawing region. If you click on **Step**, the mode will change from run mode (where the pieces update continuously) to step mode (where an update is only done when you click on **Step**). Following **Gen** is a number indicating the number of generations that have occured. The button marked **Find** will scroll so that at least one piece is in view. This is useful when all the pieces dissappear from view. The button marked **Clear** will clear the drawing region, but leave the other controls unchanged. **Reset** will reset all the panel controls, but will not erase any of the pieces, and **Quit** causes the tool to exit. The second row contains two sliders. The first controls the update speed when in run mode, the second controls the size of the pieces.

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(7)	map of ASCII character set
eqnchar(7)	special character definitions for eqn
hier(7)	file system hierarchy
man(7)	macros to format manual pages
me(7)	macros for formatting papers
ms(7)	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:

Decimal — Character

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	$	37	%	38	&	39	´
40	(	41	)	42	*	43	+	44	,	45	−	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	DEL

Octal — Character

000	NUL	001	SOH	002	STX	003	ETX	004	EOT	005	ENQ	006	ACK	007	BEL
010	BS	011	HT	012	NL	013	VT	014	NP	015	CR	016	SO	017	SI
020	DLE	021	DC1	022	DC2	023	DC3	024	DC4	025	NAK	026	SYN	027	ETB
030	CAN	031	EM	032	SUB	033	ESC	034	FS	035	GS	036	RS	037	US
040	SP	041	!	042	"	043	#	044	$	045	%	046	&	047	´
050	(	051	)	052	*	053	+	054	,	055	−	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	DEL

Hexadecimal — Character

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

/usr/preserve	*preserves editor files from crashes*
/usr/sccs	*sccs programs*
/usr/spool	*delayed execution files*
/usr/spool/mail	*system mailboxes*
/usr/spool/lpd	*printer queue(s)*
/usr/tmp	*temporary files*
/usr/ucb	*programs developed at U.C. Berkeley*
/usr/ucb/Mail	
/usr/ucb/biff	
/usr/ucb/ccat	
/usr/ucb/checknr	
/usr/ucb/chsh	
. . .	

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

man – macros to format Reference Manual pages

SYNOPSIS

nroff –man *filename* ...

troff –man *filename* ...

DESCRIPTION

These macros are used to lay out the reference pages in this manual.

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)

The –man Macro Package, in *Formatting Documents on the Sun Workstation.*

REQUESTS

Request	Cause Break	If no Argument	Explanation
.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 s d f m*	yes	-	Begin page named *n* of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets 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

CONVENTIONS

A typical manual page for a command or function is laid out as follows:

.TH TITLE [1-8]

The name of the command or function in upper-case, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME name (or comma-separated list of names) – one-line summary

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no *troff*(1) commands or escapes, and no macro requests. It is used to generate the *whatis*(1) database.

.SH SYNOPSIS

Commands:

The syntax of the command and its arguments as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with text that you supply. Syntactic symbols appear in roman face:

[ ] An argument, when surrounded by brackets is optional.

| Arguments separated by a vertical bar are exclusive. You can supply only item from such a list.

... Arguments followed by an elipsis can be repeated. When an elipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions:

If required, the data declaration, or **#include** directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

.SH DESCRIPTION

A narrative description of the command or function in detail, including how it interacts with files or data, and how it handles the standard input, standard output and standard error.

Filenames, and references to commands or functions described elswhere in the manual, are italicised. The names of options, variables and other literal terms are in boldface.

.SH OPTIONS

The list of options along with a description of how each affects the commands operation.

.SH FILES

A list of files associated with the command or function.

.SH "SEE ALSO"

A comma-separated list of related manual pages, followed by references to other published materials. This section contains no *troff*(1) escapes or commands, and no macro requests.

.SH DIAGNOSTICS

A list of diagnostic messages and an explanation of each.

.SH BUGS

A description of limitations, known defects, and possible problems associated with the command or function.

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), nroff(1), troff(1), refer(1), tbl(1)

The –me Macro Package, in *Formatting Documents on the Sun Workstation.*

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 Value	Cause Break	Explanation
.(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.

c is a controller number,0 if there only one controller for the indicated type of device.

u is a unit number, **0** if only one driver. (Over the network, the unit number is the *host* portion of the file server's IP address).

p designates a partition. The value you supply is *exclusive-or*'d with that of the default partition (indicated by the selected boot program, see *Booting a Sun-3 Over the Network*, above).

filename

is the name of a standalone program in the selected partition, such as *stand/diag* or *vmunix*. Note that *filename* is relative to the root of the selected device and partition. It never begins with /. If *filename* is not given, the boot program uses a default value (normally *vmunix*). This is stored in the "vmunix" variable in the *boot* executable file supplied by Sun, but can be patched to indicate another standalone program loaded using *adb*(1).

–a prompt interactively for the device and name of the file to boot.

boot-flags

The boot program passes all *boot-flags* to the kernel or standalone program. They are typically arguments to that program or, as with those listed below, arguments to progams that it invokes.

–b Pass the **–b** flag through the kernel to *init*(8) to skip execution of the *rc.local* script.

–h Halt after loading UNIX.

–s Pass the –s flag through the kernel to *init*(8) for single-user operation.

FILES

/boot	the standalone boot program
/tftpboot/????????	a symbolic link to the boot program for a client
/tftpboot/ndboot.sun3.pub[01]	programs to boot from a public *nd* partition
/tftpboot/ndboot.sun3.private	program to boot from a private *nd* partition
*/tftpboot/ndboot.sun2.**	Sun-2 boot programs (currently unusable)
/usr/mdec/installboot	script to install boot blocks from a remote host

SEE ALSO

init(8), kadb(8S), monitor(8s), rc(8), reboot(8)

System Administration for the Sun Workstation

Installing UNIX on the Sun Workstation

BUGS

On the Sun-2, the PROM passes in the default name "vmunix", overriding the the boot program's patchable default.

NAME

catman – create the cat files for the manual

SYNOPSIS

/usr/etc/catman [**–p**] [**–n**] [**–w**] [**–t**] [**–M** directory] [**–T** tmac.an] [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 **whatis** database.

If there is one parameter not starting with a '–', it is taken to be a list of manual sections to look in. For example

catman 123

only updates manual sections 1, 2, and 3.

If an unformatted source file contains only a line of the form ".so manx/yyy.x", a symbolic link is made in the catx or fmtx directory to the appropriate preformatted manual page. This feature allows easy distribution of the preformatted manual pages among a group of associated machines with *rdist*(1), since it makes the directories of preformatted manual pages self-contained and independent of the unformatted entries.

OPTIONS

–n Do not (re)create the **whatis** database.

–p Print what would be done instead of doing it.

–w Only create the **whatis** database. No manual reformatting is done.

–t Create *troff*ed entries in the appropiate *fmt* subdirectories instead of *nroff*ing into the *cat* subdirectories.

–M update manual pages located in the specified *directory* (*/usr/man* by default).

–T Use *tmac.an* in place of the standard manual page macros.

FILES

/usr/man	default manual directory location
/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted *nroff*ed manual pages
/usr/man/fmt?/*.*	preformatted *troff*ed manual pages
/usr/man/whatis	**whatis** database location
/usr/lib/makewhatis	command script to make whatis database

ENVIRONMENT

TROFF The name of the formatter to use when the **–t** flag is given. If not set, '**troff**' is used.

DIAGNOSTICS

man?/xxx.? (.so'ed from man?/yyy.?): No such file or directory

The file outside the parentheses is missing, and is referred to by the file inside them.

target of .so in man?/xxx.? must be relative to /usr/man

catman only allows references to filenames that are relative to the directory */usr/man*.

opendir:man?: No such file or directory

A harmless warning message indicating that one of the directories *catman* normally looks for is missing.

.: No such file or directory

A harmless warning message indicating *catman* came across an empty directory.

SEE ALSO

man(1), whatis(1)

NAME

dump, rdump – incremental file system dump

SYNOPSIS

/etc/dump [*options* [*arguments*]] *filesystem*

DESCRIPTION

Dump backs up all files in *filesystem*, or files changed after a certain date, to magnetic tape. *Options* is a string that specifies *dump* options, as shown below. Any *arguments* supplied for specific options are given as subsequent words on the command line, in the same order as that of the *options* listed.

If no *options* are given, the default is **9u**.

OPTIONS

0–9 The "dump level." All files in the *filesystem* that have been modified since the last *dump* at a lower dump level are copied to the tape. For instance, if you did a "level 2" dump on Monday, followed by a "level 4" dump on Tuesday, a subsequent "level 3" dump on Wednesday would contain all files modified or added since the "level 2" (Monday) backup. A "level 0" dump copies the entire filesystem to tape.

b *factor* Blocking factor. Specifies the blocking factor for tape writes. The default is 10 blocks per write. Note that a tape block is 1024 bytes in size, or twice the size of a disk block. The highest blocking factor available with some 6250bpi tape drives is 126.

c Cartridge. Use a cartridge instead of the standard half-inch reel. This sets the density to 1000bpi and the length to 1700 feet. When dumping to a high-density (9-track) cartridge, include the **s** (size) option with the 3825 (feet) argument to properly fill each cartridge. (This option is incompatible with the **d** option, unless you specify a density of 1000bpi with that option).

d *bpi* Tape density. The density of the tape, expressed in BPI, is taken from *bpi*. This is used to keep a running tab on the amount of tape used per reel. The default density is 1600. Unless a higher density is specified explicitly, *dump* uses its default density—even if the tape drive is capable of higher-density operation (for instance, 6250bpi).

f *dump-file*

Dump file. Use *dump-file* as the file to dump to, instead of */dev/rmt8*. If *dump-file* is specified as '–', dump to the standard output. If the filename argument is of the form *machine:device*, dump to a remote machine. Since *dump* is normally run by *root*, the name of the remote machine must appear in the *.rhosts* file of the local machine. If *dump* is called as *rdump*, the tape defaults to **dumphost:/dev/rmt8**. To direct the output to a desired remote machine, set up an alias for **dumphost** in the file */etc/hosts*.

n Notify. When this option is specified, if *dump* requires attention, it sends a terminal message (similar to *wall*(1)) to all operators in the "operator" group.

s *size* Specify the *size* of the tape or cartridge in feet. When the specified size is reached, *dump* waits for you to change the reel or cartridge. The default size is 2300 feet, except when **c** (cartridge) is specified, in which case the default is 1700. To estimate the size for a tape or cartridge of a non-standard length, use the formula:

(*length* * *tracks*) * .9

u Update the dump record. Add an entry to the file */etc/dumpdates*, for each filesystem successfully dumped that includes the filesystem name, date, and dump level. This file can be edited by the super-user.

w List the filesystems that need backing up. This information is gleaned from the files */etc/dumpdates* and */etc/fstab*. When the **w** option is used, all other options are ignored. After reporting, *dump* exits immediately.

W Like **w**, but includes all filesystems that appear in */etc/dumpdates*, along with information about their most recent dump dates and levels. Filesystems that need backing up are highlighted.

Operator Intervention

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** option, *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", as appropriate.

Since backing up a disk can involve a lot of time and effort, *dump* checkpoints 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 a defective tape has been rewound and replaced.

dump reports periodically, and in verbose fashion. Each report includes estimates of the percentage of the dump completed and how long it will take to complete the dump.

Suggested Dump Schedule

It is vital to perform full, "level 0", dumps at regular intervals. When performing a full dump, bring the machine down to single-user mode using *shutdown*(8). While preparing for a full dump, it is a good idea to clean the drive and heads.

Incremental dumps allow for convenient backup and recovery on a more frequent basis of active files, with a minimum of tape and time. However there are some tradeoffs. First, the interval between backups should be kept to a minimum (once a day at least). To guard against data loss as a result of a media failure (a rare, but possible occurrence), it is a good idea to capture active files on (at least) two dump tapes. Another consideration is the desire to keep unnecessary duplication of files to a minimum to save both operator time and tape storage. A third consideration is the ease with which a particular backed-up version of a file can be located and restored. The following four-week schedule offers a reasonable tradeoff between these goals.

	Sun	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*
Week 1:	**Full**	**5**	**5**	**5**	**5**	**3**
Week 2:		**5**	**5**	**5**	**5**	**3**
Week 3:		**5**	**5**	**5**	**5**	**3**
Week 4:		**5**	**5**	**5**	**5**	**3**

Although the Tuesday—Friday incrementals contain "extra copies" of files from Monday, this scheme assures that any file modified during the week can be recovered from the previous day's incremental dump.

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

While running, *dump* emits many verbose messages.

Exit Codes

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 tape 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.

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

edquota – edit user quotas

SYNOPSIS

/usr/etc/edquota [**–p** *proto-user*] *users...*
/usr/etc/edquota –t

DESCRIPTION

Edquota is a quota editor. One or more users may be specified on the command line. For each user a temporary file is created with an ASCII representation of the current disk quotas for that user and an editor is then invoked on the file. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, *edquota* reads the temporary file and modifies the binary quota files to reflect the changes made.

The editor invoked is *vi*(1) unless the EDITOR environment variable specifies otherwise.

Only the super-user may edit quotas. (In order for quotas to be estabished on a file system, the root directory of the file system must contain a file, owned by root, called *quotas*. See *quotaon*(1) for details.)

OPTIONS

–p duplicate the quotas of the prototypical user specified for each user specified. This is the normal mechanism used to initialize quotas for groups of users.

–t edit the soft time limits for each file system. If the time limits are zero, the default time limits in *<ufs/quota.h>* are used. Time units of sec(onds), min(utes), hour(s), day(s), week(s), and month(s) are understood. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems

SEE ALSO

quota(1), quotactl(2), quotacheck(8), quotaon(8), repquota(8)

BUGS

The format of the temporary file is inscrutible.

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 that 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), init(8), rc(8), reboot(8)

NAME

fingerd – remote user information server

SYNOPSIS

/usr/etc/in.fingerd

DESCRIPTION

fingerd is a simple protocol based on RFC742 that provides an interface to the Name and Finger programs at several network sites. The program is supposed to return a friendly, human-oriented status report on either the system at the moment or a particular person in depth. There is no required format and the protocol consists mostly of specifying a single "command line".

fingerd listens for TCP requests at port 79. Once connected it reads a single command line terminated by a <CRLF> which is passed to *finger*(1). *fingerd* closes its connections as soon as the output is finished.

If the line is null (i.e. just a <CRLF> is sent) then *finger* returns a "default" report that lists all people logged into the system at that moment.

If a user name is specified (e.g. eric<CRLF>) then the response lists more extended information for only that particular user, whether logged in or not. Allowable "names" in the command line include both "login names" and "user names". If a name is ambiguous, all possible derivations are returned.

SEE ALSO

finger(1)

BUGS

Connecting directly to the server from a TIP or an equally narrow-minded TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation. *fingerd* should be taught to filter out IAC's and perhaps even respond negatively (IAC WON'T) to all option commands received.

NAME

fparel - Sun FPA online reliability tests

SYNOPSIS

fparel [**−p***n*] [**−v**]

DESCRIPTION

fparel is a command to execute the Sun FPA online confidence and reliability test program. *fparel* tests about 90% of the functions of the FPA board, and tests all FPA contexts not in use by other processes. *fparel* runs under UNIX without disturbing other processes that may be using the FPA. *fparel* can only be run by the super-user.

After a successful pass, *fparel* writes

time, date: Sun FPA Passed. The contexts tested are: 0, 1, ... 31

to the file */usr/adm/diaglog*.

If a pass fails, *fparel* writes

time, date: Sun FPA failed

along with the test name and context number that failed, to the file */usr/adm/diaglog*. *fparel* then broadcasts the message

time, date: Sun FPA failed, disabled, service required

to all users of the system. Next, *fparel* causes the kernel to disable the FPA. Once the kernel disables the FPA, the system must be rebooted to make it accessible.

The file */etc/rc.local* should contain an entry to cause *fparel* to be invoked upon reboot to be sure that the FPA remains unaccessible in cases where rebooting doesn't correct the problem. See rc(8).

/usr/lib/crontab should contain an entry indicating that *cron*(8) is to run *fparel* daily, such as:

7 2 * * * /usr/etc/fpa/fparel

which causes *fparel* to run at seven minutes past two, every day. See *cron*(8) and *crontab*(5) for details.

OPTIONS

−p*n* Perform *n* passes. Default is *n*=1. **−p0** means perform 2147483647 passes.

−v Run in verbose mode with detailed test results to standard output.

FILES

/usr/adm/diaglog Log of *fparel* diagnostics.

NAME

icheck – file system storage consistency check

SYNOPSIS

/usr/etc/icheck [**–s**] [**–b** numbers] [filesystem]

DESCRIPTION

Note: *Icheck* has been superceded 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

SYNOPSIS

/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* is used at boot time to define the network address of each interface present on a machine. You can use *ifconfig* at a later time to redefine an interface's address or other operating parameters. 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 — The hardware Ethernet address of a given machine. The address is a six-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 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. Use the option only when trying to talk to a device that does not support ARP.

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.

PARAMETERS

The following parameters may be set with *ifconfig :*

up — Marks an interface "up." You can use it to enable an interface after an "ifconfig down." It happens automatically when you set the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

down — Marks an interface "down." When an interface is marked down, the system does not attempt to transmit messages through that interface. If possible, the interface is reset to disable reception, as well. This action does not automatically disable routes using the interface.

trailers — Enables the use of a "trailer" link level encapsulation when sending messages (the default). If a network interface supports *trailers*, the system, when possible, encapsulates outgoing messages in a manner that minimizes the number of memory–to–memory copy operations performed by the receiver. This feature is machine-dependent, and therefore not recommended.

-trailers — Disables the use of a "trailer" link level encapsulation.

arp — Enables 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 addresses and 10Mb/s Ethernet addresses.

-arp — Disables the use of the Address Resolution Protocol.

netmask *mask* — Specifies how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. You can specify the mask as a single hexadecimal number with a leading 0x, with a dot-notation Internet

address, or with a pseudo-network name listed in the network table *networks*(5). The mask contains 1's for the bit positions in the 32-bit address that are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

broadcast *address* Specifies the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 0's.

EXAMPLE

If your workstation is not attached to an Ethernet, you should mark down the **ie0** interface as follows:

ifconfig ie0 down

DIAGNOSTICS

Messages indicating that the specified interface does not exist, that the requested address is unknown, or that a user without proper privileges tried to alter an interface's configuration.

SEE ALSO

rc(8), intro(3N), netstat(8C)

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

kadb – adb-like kernel and standalone-program debugger

SYNOPSIS

> b kadb [**–d**] [*boot-flags*]

DESCRIPTION

kadb is an interactive debugger that is similar in operation to *adb*(1), and runs as a standalone program under the PROM monitor. You can use *kadb* to debug the UNIX kernel, or to debug any standalone program.

Unlike *adb*, *kadb* runs in the same supervisor virtual address space as the program being debugged — although it maintains a separate context. The debugger runs as a *coprocess* that cannot be killed (no **:k**) or rerun (no **:r**). There is no signal control (no **:i**, **:t**, or **$i**), although the UNIX keyboard facilities (^C,^S and ^Q) are simulated.

While the kernel is running under *kadb*, the abort sequence (L1-A or BREAK) causes UNIX to drop into *kadb* for debugging — as will a system panic. When running other standalone programs under *kadb* , the abort sequence will pass control to the PROM monitor. *kadb* is then invoked from the monitor by jumping to the starting address for *kadb* found in *<debug/debug.h>* (currently this can be done for both Sun-2 and Sun-3 machines with the monitor command **g fd00000**). *kadb*'s user interface is similar to *adb*. Note that *kadb* prompts with

kadb>

Most *adb* commands function in *kadb* as expected. Typing an abort sequence in response to the prompt returns you to the PROM monitor, from which you can examine control spaces that aren't accessible within *adb* or *kadb*. The PROM monitor command **c** will return control to *kadb*. As with "adb **–k**", **$p** works when debugging UNIX kernels (by actually mapping in new user pages). The verbs **?** and **/** are equivalent in *kadb* , since there is only one address space in use.

OPTIONS

kadb is booted from the PROM monitor as a standalone program. If you omit the **–d** flag, *kadb* automatically loads and runs *vmunix* from the filesystem *kadb* was loaded from. The *kadb* "vmunix" variable can be patched to change the default program to be loaded.

–d Interactive startup. Prompts with

kadb:

for a file to be loaded. >From here, you can enter a boot sequence line to load a standalone program. Boot flags entered in response to this prompt are included with those already set and passed to the program. If you type a carriage return only, *kadb* loads *vmunix* from the filesystem that *kadb* was loaded from.

boot-flags

You can specify boot flags as arguments when invoking *kadb*. Note that *kadb* always sets the **–d** (debug) boot flag, and passes it to the program being debugged.

USAGE

Refer to *adb* in *Program Debugging Tools for the Sun Workstation* .

Kernel Macros

As with *adb*, kernel macros are supported. With *kadb*, however, the macros are compiled into the debugger itself, rather than being read in from the filesystem. The *kadb* command **$M** lists macros known to *kadb*.

Setting Breakpoints

Self-relocating programs such as the Sun-3 kernel need to be relocated before breakpoints can be used. To set the first breakpoint for such a program, start it with **:s**; *kadb* is then entered after the program is relocated (when UNIX initializes its interrupt vectors). Thereafter, **:s** single-steps as with *adb*. Otherwise, use **:c** to start up the program.

Automatic Rebooting with Kadb

You can set up your workstation to automatically reboot *kadb* by patching the "vmunix" variable in */boot* with the string "kadb" instead of "vmunix". (Refer to *adb* in *Program Debugging Tools for the Sun Workstation* for details on how to patch executables.)

Kadb on a Diskless Workstation

If your workstation is set up to boot over the network from a partition other than *pub0*, then you should patch the short *kadb* variable "ndbootdev" to be "0x0", for the *private nd* partition, or "0x41", for the *pub1 nd* partition. This will insure that the file to be debugged and *kadb* come from the same nd filesystem.

If "ndbootdev" is not patched, then you must be explicit when booting with *kadb*. Use the command

> b kadb -d

so that *kadb* will prompt for the program to be debugged. At the prompt use the commmand

kadb: *device(, , p)filename*

where *p* is "0x1" for the *pub1 nd* partition or "0x40" for the *private nd* partition. Note that these values for *p* (partition) will work if the file to be debugged is in the same filesystem as *kadb*.

FILES

/vmunix
/boot
/kadb
/usr/include/debug/debug.h

SEE ALSO

adb(1), *boot*(8S)
Program Debugging Tools for the Sun Workstation
Writing Device Drivers for the Sun Workstation

BUGS

There is no floating-point support.

kadb cannot reliably single-step over instructions that change the status register.

When sharing the keyboard with UNIX the monitor's input routines can leave the keyboard in a confued state. If this should happen, disconnect the keybooard momentarily and then reconnect it. This forces the keyboard to reset as well as initiating an abort sequence.

Most of the bugs listed in *adb*(1) also apply to *kadb*.

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

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*(1V).

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

L1 -a

BREAK

DESCRIPTION

The CPU board of the Sun workstation contains a PROM (or set of PROMs), called the *monitor*, that controls the system during startup. The monitor tests the system and then searches for and attempts to boot UNIX. If you interrupt the boot procedure, or by typing either **L1–a** or **BREAK**, it issues the prompt:

>

and accepts commands interactively.

COMMANDS

A[*n*] [*action* ...]

open **A**-register (cpu address register) *n*, and perform indicated actions. *n* can be from **0** to **7**. The default is **0**. *action* is a data value in hex; a non-hex character terminates command input.

B [*device* [(*c*,*u*,*p*)]] [*pathname*]

boot. Resets appropriate parts of the system, then bootstraps. This allows bootstrap loading of programs from various devices (such as a disk, tape, or Ethernet connection).

device is one of:

ie	Intel Ethernet
le	Lance Ethernet
sd	SCSI disk
st	SCSI 1/4" tape
mt	Tape Master 9-track 1/2" tape
xt	Xylogics 1/2" tape
xy	Xylogics 440/450 disk

c is a controller number (**0** if only one controller),

u is a unit number (**0** if only one driver), and,

p is a partition.

pathname
is a pathname for a program such as */stand/diag*. **/vmunix** is the default.

B with no arguments will cause a default boot, either from the disk, or from the Ethernet controller.

B? displays all boot devices and their device arguments.

C [*addr*]

continue a program. When given, *addr* is the address at which execution will begin. The default is the current PC. Registers are restored to the values shown by **A**, **D**, and **R** commands.

D [*n*] [*action* ...]

open **D**-register (cpu data register) *n*, and perform indicated actions. *n* can be from **0** to **7**. The default is **0**.

E [*addr*] [*param*]

open the 16 bit word at *addr* (default zero) in the address space defined by the **S** command.

F *addr1 addr2* [*pattern*] [size] *Sun 3 only.*

Fill address space from (lower) *addr1* to (higher) *addr2* with the constant, *pattern*, specified by *size*:

b byte format (the default),

w word format, or

l long word format.

For example, the following command fills the address block from 0x1000 to 0x2000 with the word pattern, 0xABCD:

F 1000 2000 ABCD W

G [*addr*]

Start the program by executing a subroutine call to the address *addr* if given, or else to the current PC. The values of the address and data registers are undefined. The status register is set to 0x2700.

G0 When the monitor is running as a result of being interrupted, force a panic and produce a crash dump.

G4 When the monitor is running as a result of being interrupted, force a kernel stack trace.

H *Sun 3 only*

Display the menu of monitor commands, and their descriptions.

K [*number*]

If number is:
0 cpu reset only. This is the default.

1 reset cpu and mmu.

2 reboot. Resets and clears memory, as with a power-on reset. Runs the PROM-based diagnostic self test, which can take from 5 to 180 seconds depending upon how much memory is being tested.

KB display the system banner.

L [*addr*] [*actions*]

open the long (32 bit) word at memory address *addr* (default zero) in the address space defined by the S command (below).

M [*addr*] [*actions*]

open the segment map entry that maps virtual address *addr* (default zero) in the address space defined by the S command (below). The segment map address is the virtual address field from address bit 27 thru bit 17 of the virtual address presented by the cpu to the mmu.

O [*addr*] [*actions*]

open the byte location specified by *addr* (default zero) in the address space defined by the S command below.

P [*addr*] [*actions*]

open the page map entry that maps virtual address *addr* (default zero) in the address space defined by the S command.

Q [*addr*] [*actions*] *Sun 3 only.*

open the EEPROM address *addr* (default zero) in the EEPROM address space. All addresses are referenced from the beginning or base of the EEPROM in physical address space, and a limit check is performed to insure that no address beyond the EEPROM physical space is accessed. This command is used to examine/modify configuration parameters specifying such things as amount of memory to test during self test, whether to display a standard or custom banner, if a serial port (A or B) is to be the system console, etc.

R [*reg*] [*actions*]
open the miscellaneous registers. *reg* can be one of:

CA 68020 Cache Address Register

CC 68020 Cache Control Register

CX 68020 System and User Context

DF Destination Function code

IS 68020 Interrupt Stack Pointer

MS 68020 Master Stack Pointer

PC Program Counter

SC 68010 System Context

SF Source Function code

SR Status Register

SS 68010 Supervisor Stack Pointer

UC 68010 User Context

US User Stack Pointer

VB Vector Base

Alterations to these registers (except **SC** and **UC**, or **CX**) do not take effect until the next **C** command.

S [*code*] set or query the address space to be used by subsequent memory access commands. *code* is one of:

0 undefined.

1 user data space.

2 user program space.

3 user control space.

4 undefined.

5 supervisor data space.

6 supervisor program space.

7 supervisor control space.

T [*command*] *Sun 3 only*

trace *command*. Works with standalone programs that do not affect interrupt vectors.

U [*arg*] manipulate the serial ports and switches the current operator I/O device. *arg* can have any of the following values (**[AB]** indicates one of **A** or **B**):

[AB] select serial port A or B as input and output device

[AB]io select serial port A or B as input and output device

[AB]i select serial port A or B for input only

[AB]o select serial port A or B for output only

k select keyboard for input

ki select keyboard for input

s select screen for output

so select screen for output

ks,sk select keyboard for input and screen for output

[AB]# set speed of serial port A (or B) to # (such as 1200,9600,..)

e echo input to output

ne don't echo input to output

u *addr* set virtual serial port address to *addr* .

If no serial port is specified when changing speeds, the current input device is changed.

At power-up, the following default settings are used: the default console input device is the Sun keyboard or if the keyboard is unavailable, serial port A. The default console output device is the Sun screen or if the graphics board is unavailable, serial port A. All serial ports are set to 9600 Baud.

V *addr1 addr2* [*size*] *Sun 3 only*

display the contents of addresses from (lower) *addr1* to (higher) address *addr2* in the format specified by *size*:

b byte format (the default),

w word format, or

l long word format.

Enter return to pause for viewing; enter another return character resume the display. To terminate the display at any time, press the space bar. Or, you can use ^S and ^Q to stop and start the display.

For example, the following command displays the contents of virtual address space from address 0x1000 to 0x2000 in word format:

V 1000 2000 W

W [*addr*] [*arg*] *Sun 3 only.*

Vector to *addr*. *arg* is one of:

print prints the contents of virtual address *addr* as a string.

dump initiates a crash dump.

trace produces a stack trace.

X *Sun 3 only*

display a menu of extended tests to be presented, with loop and print options also selectable. These test commands are provided to permit additional testing of such things as the I/O port connectors at the handle edge of the CPU board, Video memory, workstation memory and the workstation keyboard, as well as permit the boot device paths to be tested.

Z [*addr*] *Sun 3 only.*

set a breakpoint at *addr* in the address space selected by the S command.

NAME

mount, umount – mount and dismount filesystems

SYNOPSIS

/etc/mount [**–p**]
/etc/mount –a[fv] [**–t** *type*]
/etc/mount [**–frv**] [**–t** *type*] [**–o** *options*] *fsname dir*
/etc/mount [**–vf**] [**–o** *options*] *fsname* | *dir*

/etc/umount [**–t** *type*] [**–h** *host*]
/etc/umount –a[v]
/etc/umount [**–v**]

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*.

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 the file */etc/fstab* (see *fstab*(5)) for an entry whose *dir* or *fsname* field matches the given argument. For example, if this line is in */etc/fstab*:

/dev/xy0g /usr 4.2 rw 1 1

then the commands **mount /usr** and **mount /dev/xy0g** are shorthand for **mount /dev/xy0g /usr**

MOUNT OPTIONS

–p Print the list of mounted filesystems in a format suitable for use in */etc/fstab*.

–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 is mounted. Filesystems are not necessarily mounted in the order listed in */etc/fstab* .

–f Fake a new */etc/mtab* entry, but do not actually mount any filesystems.

–v Verbose — *mount* displays a message indicating the filesystem being mounted.

–t The next argument is the filesystem type. The accepted types are: **4.2**, and **nfs**; see *fstab*(5) for a description of these filesystem types.

–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 occur when access times are updated, whether or not any explicit write is attempted.

–o Specify *options* , a list of comma seperated words from the list below. Some options are valid for all filesystem types, while others apply to a specific type only.

options valid on *all* file systems (the default is **rw,suid**):

rw read/write.

ro read-only.

suid set-uid execution allowed.

nosuid set-uid execution not allowed.

noauto do not mount this file system automatically (mount -a).

options specific to **4.2** file systems (the default is **noquota**).

quota usage limits enforced.

noquota usage limits not enforced.

options specific to **nfs** (NFS) file systems (the defaults are:

fg,retry=10000,timeo=7,retrans=3,port=NFS_PORT,hard

with defaults for *rsize* and *wsize* set by the kernel):

bg if the first mount attempt fails, retry in the background.

fg retry in foreground.

retry=*n* set number times to retry mount to *n*.

rsize=*n* set read buffer size to *n* bytes.

wsize=*n* set write buffer size to *n bytes*.

timeo=*n* set NFS timeout to *n* tenths of a second.

retrans=*n* set number of NFS retransmissions to *n*.

port=*n* set server IP port number to *n*.

soft return error if server doesn't respond.

hard retry request until server responds.

intr allow keyboard interrupts on hard mounts.

The **bg** option causes *mount* to run in the background if the server's *mountd*(8) does not respond. *mount* attempts each request **retry=*n*** times before giving up. Once the filesystem is mounted, each NFS request made in the kernel waits **timeo=*n*** tenths of a second for a response. If no response arrives, the time-out is multiplied by **2** and the request is retransmitted. When **retrans=*n*** retransmissions have been sent with no reply a **soft** mounted filesystem returns an error on the request and a **hard** mounted filesystem prints a message and retries the request. Filesystems that are mounted **rw** (read-write) should use the **hard** option. The **intr** option allows keyboard interrupts to kill a process that is hung waiting for a response on a hard mounted filesystem. The number of bytes in a read or write request can be set with the **rsize** and **wsize** options.

UMOUNT OPTIONS

–h *host* Unmount all filesystems listed in */etc/mtab* that are remote-mounted from *host*.

–a Attempt to unmount all the filesystems currently mounted (listed in */etc/mtab*). 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	table of mounted filesystems
/etc/fstab	table of filesystems mounted at boot

SEE ALSO

mount(2), unmount(2), fstab(5), mountd(8C), nfsd(8C)

BUGS

Mounting filesystems full of garbage crashes the system.

No more than one ND client should mount an ND disk partition "read-write" or the file system may become corrupted.

If the directory on which a filesystem is to be mounted is a symbolic link, the filesystem is mounted on *the directory to which the symbolic link refers*, rather than being mounted on top of the symbolic link itself.

NAME

setup – Sun UNIX installation program

SYNOPSIS

setup

DESCRIPTION

setup is the program supplied by Sun to install major Sun Unix releases such as 2.0 or 3.0. *setup* allows a system administrator to install major Sun Unix release on new hardware, upgrade between major releases, and add additional hardware to existing machines.

setup provide both a tty interface for cursor addressable terminals and a SunWindows system interface for use on bit mapped displays. The *Setup Reference Manual* contains a detailed description of the use of *setup*.

Initially, *setup* asks the following questions in a menu format before entering the tty or SunWindows interface. For all menus respond to the >> prompt with the corresponding number of the menu item you choose.

The first question asked is the mode of use of *setup*.

```
Are you running setup:
      1) to install on a new system
      2) re-entrantly
      3) to upgrade an existing system
      4) in demonstration mode
>>
```

The next question is to determine the type of interface to be used. Note that the cursor addressable interface can be used within a *shelltool*(1) under SunWindows.

```
Will you be running setup from:
      1) a Sun bit mapped display device
      2) a cursor addressable terminal
>>
```

If you have selected the tty interface for cursor addressable terminals, *setup* asks for the terminal type.

```
Select your terminal type:
      1) Televideo 925
      2) Wyse Model 50
      3) Sun Workstation
      4) Other
>>
```

If you select "Other", the name of the terminal must correspond to a name in the *termcap*(5) database.

```
Enter the terminal type (your terminal type must be in /etc/termcap):
>>
```

setup begins running the interface for the terminal-type you have selected.

FILES

/etc/hosts
/etc/nd.local
/etc/ethers
/etc/rc.local
/etc/rc.boot
/etc/setup.info
/usr/lib/sendmail.cf

BUGS

setup will not run on tty devices that do not support cursor addressing and are not registered in the *termcap*(5) data base.

NAME

showmount – show all remote mounts

SYNOPSIS

/usr/etc/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

statd – network status monitor

SYNOPSIS

/etc/rpc.statd

DESCRIPTION

Statd is an intermediate version of the status monitor. It interacts with *lockd*(8c) to provide the crash and recovery functions for the locking services on NFS.

FILES

/etc/statmon/current
/etc/statmon/backup
/etc/statmon/state

SEE ALSO

lockd(8C), statmon(5)

BUGS

The crash of a site is only detected upon its recovery.

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:

1. Clear the sticky bit with *chmod*(1V).
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 once again, if still needed.

A directory for which the 'sticky bit' is set restricts deletion of files it contains. A file in a sticky directory may only be removed or renamed by a user who has write permission on the directory, and either owns the file, owns the directory, or is the super-user. This is useful for directories such as */tmp*, which must be publicly writable, but which should deny users access to arbitrarily delete or rename the files of others.

Any user may create a sticky directory. Only the super-user can set the sticky bit on a non-directory file.

UNIX Interface Reference Manual Insertion Pages

NAME

chmod, fchmod – change mode of file

SYNOPSIS

#include /usr/include/sys/stat.h

chmod(path, mode)
char *path;
int mode;

fchmod(fd, mode)
int fd, mode;

DESCRIPTION

The file whose name is given by *path* or referenced by the descriptor *fd* has its mode changed to *mode*. Modes are constructed by *or*'ing together some combination of the following:

S_ISUID	04000 set user ID on execution
S_ISGID	02000 set group ID on execution
S_ISVTX	01000 save text image after execution (sticky bit)
S_IREAD	00400 read by owner
S_IWRITE	00200 write by owner
S_IEXEC	00100 execute (search on directory) by owner
	00070 read, write, execute (search) by group
	00007 read, write, execute (search) by others

These bit patterns are defined in **/usr/include/sys/stat.h.**

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user and the process attempts to set the set group ID bit on a file owned by a group which is not in its group access list, mode bit 02000 (set group ID on execution) is cleared.

If an executable file is set up for sharing (this is the default) then mode 01000 (save text image after execution) prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. If this mode is set on a directory, an unprivileged user may not delete or rename files of other users in that directory. If the effective user ID of the process is not super-user and the object is not a directory, this bit is cleared.

If a user other than the super-user writes to a file, the set user ID and set group ID bits are turned off. This makes the system somewhat more secure by protecting set-user-ID (set-group-ID) files from remaining set-user-ID (set-group-ID) if they are modified, at the expense of a degree of compatibility.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

ERRORS

chmod will fail and the file mode will be unchanged if:

ENOTDIR A component of the path prefix of *path* is not a directory.

EINVAL *path* contains a byte with the high-order bit set.

ENAMETOOLONG
The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters.

ENOENT The file referred to by *path* does not exist.

EACCES Search permission is denied for a component of the path prefix of *path*.

ELOOP	Too many symbolic links were encountered in translating *path*.
EPERM	The effective user ID does not match the owner of the file and the effective user ID is not the super-user.
EINVAL	*fd* refers to a socket, not to a file.
EROFS	The file referred to by *path* resides on a read-only file system.
EFAULT	*path* points outside the process's allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.

fchmod will fail if:

EBADF	The descriptor is not valid.
EROFS	The file referred to by *fd* resides on a read-only file system.
EPERM	The effective user ID does not match the owner of the file and the effective user ID is not the super-user.
EIO	An I/O error occurred while reading from or writing to the file system.

FILES

/usr/include/sys/stat.h

SEE ALSO

open(2V), chown(2), stat(2), sticky(8)

NAME

getrlimit, setrlimit – control maximum system resource consumption

SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

getrlimit(resource, rlp)
int resource;
struct rlimit *rlp;

setrlimit(resource, rlp)
int resource;
struct rlimit *rlp;
```

DESCRIPTION

Limits on the consumption of system resources by the current process and each process it creates may be obtained with the *getrlimit* call, and set with the *setrlimit* call.

The *resource* parameter is one of the following:

RLIMIT_CPU	the maximum amount of cpu time (in seconds) to be used by each process.
RLIMIT_FSIZE	the largest size, in bytes, of any single file that may be created.
RLIMIT_DATA	the maximum size, in bytes, of the data segment for a process; this defines how far a program may extend its break with the *sbrk*(2) system call.
RLIMIT_STACK	the maximum size, in bytes, of the stack segment for a process; this defines how far a program's stack segment may be extended automatically by the system.
RLIMIT_CORE	the largest size, in bytes, of a *core* file that may be created.
RLIMIT_RSS	the maximum size, in bytes, to which a process's resident set size may grow. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system will prefer to take memory from processes that are exceeding their declared resident set size.

A resource limit is specified as a soft limit and a hard limit. When a soft limit is exceeded a process may receive a signal (for example, if the cpu time is exceeded), but it will be allowed to continue execution until it reaches the hard limit (or modifies its resource limit). The *rlimit* structure is used to specify the hard and soft limits on a resource,

```
struct rlimit {
        int     rlim_cur;       /* current (soft) limit */
        int     rlim_max;       /* hard limit */
};
```

Only the super-user may raise the maximum limits. Other users may only alter *rlim_cur* within the range from 0 to *rlim_max* or (irreversibly) lower *rlim_max*.

An "infinite" value for a limit is defined as RLIM_INFINITY (0x7fffffff).

Because this information is stored in the per-process information, this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to *csh*(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way: a *brk* or *sbrk* call will fail if the data space limit is reached, or the process will be killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file I/O operation which would create a file that is too large will cause a signal SIGXFSZ to be generated; this normally terminates the process, but may be caught. When the soft CPU time limit is exceeded, a signal SIGXCPU is sent to the offending process.

RETURN VALUE

A 0 return value indicates that the call succeeded, changing or returning the resource limit. A return value of −1 indicates that an error occurred, and an error code is stored in the global location *errno*.

ERRORS

The possible errors are:

EFAULT	The address specified for *rlp* is invalid.
EINVAL	An invalid *resource* was specified; or in a *setrlimit* call, the new *rlim_cur* exceeds the new *rlim_max*.
EPERM	The limit specified to *setrlimit* would have raised the maximum limit value, and the caller is not the super-user.

SEE ALSO

csh(1), quota(2)

BUGS

There should be *limit* and *unlimit* commands in *sh*(1) as well as in *csh*.

NAME

gettimeofday, settimeofday – get or set the date and time

SYNOPSIS

```
#include <sys/time.h>

gettimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;

settimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the *gettimeofday* call, and set with *settimeofday*. The current time is expressed in elapsed seconds and microseconds since, January 1, 1970 (zero hour). The resolution of the system clock is hardware dependent; the time may be updated continuously, or in "ticks."

The structures pointed to by *tp* and *tzp* are defined in *<sys/time.h>* as:

```
struct timeval {
        long    tv_sec;         /* seconds since Jan. 1, 1970 */
        long    tv_usec;        /* and microseconds */
};

struct timezone {
        int     tz_minuteswest; /* of Greenwich */
        int     tz_dsttime;     /* type of dst correction to apply */
};
```

The *timezone* structure indicates the local time zone (measured in minutes westward from Greenwich), and flag that indicates the type of Daylight Saving Time correction to apply. Note that this flag does *not* indicate whether DST is currently in effect.

If *tzp* is a zero pointer, the timezone information is not returned or set.

Only the super-user may set the time of day or the time zone.

RETURN

A −1 return value indicates an error occurred; in this case an error code is stored in the global variable *errno*. Other return codes indicate the type of Daylight Savings Time currently in effect (as defined in */usr/include/sys/time.h*):

0	DST_NONE: Daylight Savings Time not observed
1	DST_USA: United States DST
2	DST_AUST: Australian DST
3	DST_WET: Western European DST
4	DST_MET: Middle European DST
5	DST_EET: Eastern European DST
6	DST_CAN: Canadian DST
7	DST_GB: Great Britian and Eire DST
8	DST_RUM: Rumanian DST
9	DST_TUR: Turkish DST
10	DST_AUSTALT: Australian-style DST with shift in 1986

ERRORS

The following error codes may be set in *errno*:

EFAULT	An argument address referenced invalid memory.

EPERM A user other than the super-user attempted to set the time.

SEE ALSO

date(1), adjtime(2), ctime(3)

BUGS

Time is never correct enough to believe the microsecond values. There should a mechanism by which, at least, local clusters of systems might synchronize their clocks to millisecond granularity.

Daylight Savings Time correction tables aren't guaranteed to be correct for specific locales.

NAME

getuid, geteuid – get user identity

SYNOPSIS

uid = getuid()
int uid;

euid = geteuid()
int euid;

DESCRIPTION

Getuid returns the real user ID of the current process, *geteuid* the effective user ID.

The real user ID identifies the person who is logged in. The effective user ID gives the process additional permissions during execution of "set-user-ID" mode processes, which use *getuid* to determine the real-user-id of the process that invoked them.

SEE ALSO

getgid(2), setreuid(2)

NAME

read, readv – read input

SYNOPSIS

```
cc = read(d, buf, nbytes)
int cc, d;
char *buf;
int nbytes;

#include <sys/types.h>
#include <sys/uio.h>

cc = readv(d, iov, iovcnt)
int cc, d;
struct iovec *iov;
int iovcnt;
```

DESCRIPTION

read attempts to read *nbytes* of data from the object referenced by the descriptor *d* into the buffer pointed to by *buf*. *readv* performs the same action, but scatters the input data into the *iovcnt* buffers specified by the members of the *iov* array: iov[0], iov[1], ..., iov[iovcnt–1].

For *readv*, the *iovec* structure is defined as

```
struct iovec {
        caddr_t  iov_base;
        int      iov_len;
};
```

Each *iovec* entry specifies the base address and length of an area in memory where data should be placed. *readv* will always fill an area completely before proceeding to the next.

On objects capable of seeking, the *read* starts at a position given by the pointer associated with *d* (see *lseek*(2)). Upon return from *read*, the pointer is incremented by the number of bytes actually read.

Objects that are not capable of seeking always read from the current position. The value of the pointer associated with such an object is undefined.

Upon successful completion, *read* and *readv* return the number of bytes actually read and placed in the buffer. The system guarantees to read the number of bytes requested if the descriptor references a normal file which has that many bytes left before the end-of-file, but in no other case.

If the returned value is 0, then end-of-file has been reached.

When attempting to read from a descriptor associated with an empty pipe, socket, or FIFO:

If O_NDELAY is set, the read will return a –1 and *errno* will be set to EWOULDBLOCK.

If O_NDELAY is clear, the read will block until data is written to the pipe or the file is no longer open for writing.

When attempting to read from a descriptor associated with a tty that has no data currently available:

If O_NDELAY is set, the read will return a –1 and *errno* will be set to EWOULDBLOCK.

If O_NDELAY is clear, the read will block until data becomes available.

If O_NDELAY is set, and less data are available than are requested by the *read* or *readv*, only the data that are available are returned, and the count indicates how many bytes of data were actually read.

SYSTEM V DESCRIPTION

When an attempt is made to read a descriptor which is in no-delay mode, and there is no data currently available, *read* will return a 0 instead of returning a –1 and setting *errno* to EWOULDBLOCK. Note that this is indistinguishable from end-of-file.

RETURN VALUE

If successful, the number of bytes actually read is returned. Otherwise, a −1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

read and *readv* will fail if one or more of the following are true:

EBADF	*d* is not a valid file descriptor open for reading.
EISDIR	*d* refers to a directory which is on a file system mounted using the NFS.
EFAULT	*buf* points outside the allocated address space.
EIO	An I/O error occurred while reading from or writing to the file system.
EINTR	A read from a slow device was interrupted before any data arrived by the delivery of a signal.
EINVAL	The pointer associated with *d* was negative.
EWOULDBLOCK	The file was marked for non-blocking I/O, and no data were ready to be read. In addition, *readv* may return one of the following errors:
EINVAL	*Iovcnt* was less than or equal to 0, or greater than 16.
EINVAL	One of the *iov_len* values in the *iov* array was negative.
EINVAL	The sum of the *iov_len* values in the *iov* array overflowed a 32-bit integer.
EFAULT	Part of *iov* points outside the process's allocated address space.

SEE ALSO

dup(2), fcntl(2), open(2), pipe(2), select(2), socket(2), socketpair(2)

RETURN VALUE

Upon successful completion, a non-negative integer, namely a shared memory identifier is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

intro(2), shmctl(2), shmop(2)

NAME

shmop, shmat, shmdt – shared memory operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat (shmid, shmaddr, shmflg)
int shmid;
char *shmaddr
int shmflg;

int shmdt (shmaddr)
char *shmaddr
```

DESCRIPTION

shmat attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the data segment of the calling process. The segment is attached at the address specified by one of the following criteria:

If *shmaddr* is equal to zero, the segment is attached at the first available address as selected by the system.

If *shmaddr* is not equal to zero and (*shmflg* & SHM_RND) is "true", the segment is attached at the address given by (*shmaddr* - (*shmaddr* modulus SHMLBA)).

If *shmaddr* is not equal to zero and (*shmflg* & SHM_RND) is "false", the segment is attached at the address given by *shmaddr*.

The segment is attached for reading if (*shmflg* & SHM_RDONLY) is "true" {READ}, otherwise it is attached for reading and writing {READ/WRITE}.

shmdt detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*.

ERRORS

shmat will fail and not attach the shared memory segment if one or more of the following are true:

EINVAL	*Shmid* is not a valid shared memory identifier.
EACCES	Operation permission is denied to the calling process (see *intro*(2)).
ENOMEM	The available data space is not large enough to accommodate the shared memory segment.
EINVAL	*shmaddr* is not equal to zero, and the value of (*shmaddr* - (*shmaddr* modulus SHMLBA)) is an illegal address.
EINVAL	*shmaddr* is not equal to zero, (*shmflg* & SHM_RND) is "false", and the value of *shmaddr* is an illegal address.
EMFILE	The number of shared memory segments attached to the calling process would exceed the system-imposed limit.

shmdt will fail and not detach the shared memory segment if:

EINVAL	*shmaddr* is not the data segment start address of a shared memory segment.

RETURN VALUES

Upon successful completion, the return value is as follows:

shmat returns the data segment start address of the attached shared memory segment.

shmdt returns a value of 0.

Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

exec(2), exit(2), fork(2), intro(2), shmctl(2), shmget(2).

NAME

shutdown – shut down part of a full-duplex connection

SYNOPSIS

shutdown(s, how)
int s, how;

DESCRIPTION

The *shutdown* call causes all or part of a full-duplex connection on the socket associated with *s* to be shut down. If *how* is 0, then further receives will be disallowed. If *how* is 1, then further sends will be disallowed. If *how* is 2, then further sends and receives will be disallowed.

DIAGNOSTICS

A 0 is returned if the call succeeds, –1 if it fails.

ERRORS

The call succeeds unless:

EBADF	*S* is not a valid descriptor.
ENOTSOCK	*S* is a file, not a socket.
ENOTCONN	The specified socket is not connected.

SEE ALSO

connect(2), socket(2)

BUGS

The *how* values should be defined constants.

NAME

socket – create an endpoint for communication

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>

s = socket(af, type, protocol)
int s, af, type, protocol;

DESCRIPTION

Socket creates an endpoint for communication and returns a descriptor.

The *af* parameter specifies an address format with which addresses specified in later operations using the socket should be interpreted. These formats are defined in the include file *<sys/socket.h>*. The currently understood formats are

AF_UNIX	(UNIX path names),
AF_INET	(ARPA Internet addresses),
AF_PUP	(Xerox PUP-I Internet addresses), and
AF_IMPLINK	(IMP "host at IMP" addresses).

The socket has the indicated *type* which specifies the semantics of communication. Currently defined types are:

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
SOCK_SEQPACKET
SOCK_RDM

A SOCK_STREAM type provides sequenced, reliable, two-way connection based byte streams with an out-of-band data transmission mechanism. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). SOCK_RAW sockets provide access to internal network interfaces. The types SOCK_RAW, which is available only to the super-user, and SOCK_SEQPACKET and SOCK_RDM, which are planned, but not yet implemented, are not described here.

The *protocol* specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type using a given address format. However, it is possible that many protocols may exist in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see *services*(5) and *protocols*(5).

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a *connect*(2) call. Once connected, data may be transferred using *read*(2V) and *write*(2V) calls or some variant of the *send*(2) and *recv*(2) calls. When a session has been completed a *close*(2) may be performed. Out-of-band data may also be transmitted as described in *send*(2) and received as described in *recv*(2).

The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with −1 returns and with ETIMEDOUT as the specific code in the global variable errno. The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (e.g. 5 minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in *send*(2) calls. It is also possible to receive datagrams at such a socket with *recv*(2).

An *fcntl*(2) call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives.

The operation of sockets is controlled by socket level *options*. These options are defined in the file *<sys/socket.h>* and explained below. *Setsockopt* and *getsockopt*(2) are used to set and get options, respectively.

SO_DEBUG	turn on recording of debugging information
SO_REUSEADDR	allow local address reuse
SO_KEEPALIVE	keep connections alive
SO_DONTROUTE	do no apply routing on outgoing messages
SO_LINGER	linger on close if data present
SO_DONTLINGER	do not linger on close

SO_DEBUG enables debugging in the underlying protocol modules. SO_REUSEADDR indicates the rules used in validating addresses supplied in a *bind*(2) call should allow reuse of local addresses. SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address. SO_LINGER and SO_DONTLINGER control the actions taken when unsent messags are queued on socket and a *close*(2) is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the *close* attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the *setsockopt* call when SO_LINGER is requested). If SO_DONTLINGER is specified and a *close* is issued, the system will process the close in a manner which allows the process to continue as quickly as possible.

RETURN VALUE

A −1 is returned if an error occurs, otherwise the return value is a descriptor referencing the socket.

ERRORS

The *socket* call fails if:

EAFNOSUPPORT	The specified address family is not supported in this version of the system.
ESOCKTNOSUPPORT	The specified socket type is not supported in this address family.
EPROTONOSUPPORT	The specified protocol is not supported.
EMFILE	The per-process descriptor table is full.
ENOBUFS	No buffer space is available. The socket cannot be created.
EPROTOTYPE	The protocol is the wrong type for the socket.

SEE ALSO

accept(2), bind(2), connect(2), getsockname(2), getsockopt(2), ioctl(2), listen(2), recv(2), select(2), send(2), shutdown(2), socketpair(2)

Inter-Process Communication Primer in *Networking on the Sun Workstation*

BUGS

The use of keepalives is a questionable feature for this layer.

NAME

intro – introduction to library functions

DESCRIPTION

Section 3 describes library routines. The main C library is */lib/libc.a*, which contains all system call entry points described in section 2, as well as functions described in several subsections here. The primary functions are described in the main section 3. Functions associated with the "standard I/O library" used by many C programs are found in section 3S. The main C library also includes Internet network functions, described in section 3N, and routines providing compatibility with other UNIX systems, described in section 3C.

Other sections are:

(3F) This section, for FORTRAN library routines and functions, is contained in the *FORTRAN Programmer's Guide*.

(3M) The Math Library. C declarations for the types of functions are be obtained from the include file *<math.h>*. To use these functions with C programs compile them with the **–lm** option with *cc*(1). They are automatically loaded as needed by the FORTRAN and Pascal compilers *f77*(1) and *pc*(1).

(3V) The System V Compatibility Library. System V versions of functions that are not yet merged into the standard Sun libraries. To use these functions, compile programs with */usr/5bin/cc* , instead of */bin/cc*.

(3X) Various specialized libraries have not been given distinctive captions. Files in which such libraries are found are named on appropriate pages if they don't appear in the *libc* library.

FILES

/lib/libc.a	C Library ((2), (3), (3N) and (3C) routines)
/usr/lib/libc_p.a	Profiling C library (for *gprof*(1))
/usr/lib/libm.a	Math Library **–lm** (see section 3M)
/usr/lib/libm_p.a	Profiling version of **–lm**
/usr/lib/libcurses.a	screen management routines (see *curses*(3X)
/usr/lib/libdbm.a	data base management routines (see *dbm*(3X))
/usr/lib/libmp.a	multiple precision math library (see *mp*(3X))
/usr/lib/libtermcap.a	terminal handling routines (see *termcap*(3X))
/usr/lib/libtermcap_p.a	"
/usr/lib/libtermlib	(link to /usr/lib/libtermcap.a)
/usr/lib/libtermlib_p.a	(link to /usr/lib/libtermcap_p.a)
/usr/lib/libplot*.a	plot routines (see *plot*(3X))

SEE ALSO

intro(3C), intro(3S), intro(3F), intro(3M), intro(3N), nm(1), ld(1), cc(1), f77(1), intro(2)

DIAGNOSTICS

Functions in the math library (section 3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases the external variable *errno* (see *intro*(2)) is set to the value EDOM (domain error) or ERANGE (range error). The values of EDOM and ERANGE are defined in the include file *<errno.h>*.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
a64l	a64l(3)	convert base-64 ASCII to long
abort	abort(3)	generate a fault
abs	abs(3)	integer absolute value
acos	sin(3M)	trigonometric functions
acosh	asinh(3M)	inverse hyperbolic function
addmntent	getmntent(3)	get file system descriptor file entry
alarm	alarm(3C)	schedule signal after specified time

alloca	malloc(3)	memory allocator
alphasort	scandir(3)	scan a directory
asctime	ctime(3)	convert date and time to ASCII
asin	sin(3M)	trigonometric functions
asinh	asinh(3M)	inverse hyperbolic function
assert	assert(3)	program verification
atan	sin(3M)	trigonometric functions
atanh	asinh(3M)	inverse hyperbolic function
atof	atof(3)	convert ASCII to numbers
atoi	atof(3)	convert ASCII to numbers
atol	atof(3)	convert ASCII to numbers
bcmp	bstring(3)	bit and byte string operations
bcopy	bstring(3)	bit and byte string operations
bsearch	bsearch(3)	binary search a sorted table
bzero	bstring(3)	bit and byte string operations
cabs	hypot(3M)	Euclidean distance
calloc	malloc(3)	memory allocator
cbc_crypt	des_crypt(3)	fast DES encryption
cbrt	sqrt(3M)	cube root
ceil	floor(3M)	ceiling
cfree	malloc(3)	memory allocator
clearerr	ferror(3S)	stream status inquiries
clock	clock(3C)	report CPU time used
closedir	directory(3)	directory operations
closelog	syslog(3)	control system log
copysign	ieee(3M)	copysign remainder exponent manipulations
cos	sin(3M)	trigonometric functions
cosh	sinh(3M)	hyperbolic functions
crypt	crypt(3)	DES encryption
ctermid	ctermid(3S)	generate filename for terminal
ctime	ctime(3)	convert date and time to ASCII
cuserid	cuserid(3S)	get character login name of user
des_crypt	des_crypt(3)	fast DES encryption
des_setparity	des_crypt(3)	fast DES encryption
drand48	drand48(3)	generate uniformly distributed pseudo-random numbers
drem	ieee(3M)	copysign remainder exponent manipulations
dysize	ctime(3)	convert date and time to ASCII
ecb_crypt	des_crypt(3)	fast DES encryption
ecvt	ecvt(3)	output conversion
edata	end(3)	last locations in program
encrypt	crypt(3)	DES encryption
end	end(3)	last locations in program
endfsent	getfsent(3)	get file system descriptor file entry
endgrent	getgrent(3)	get group file entry
endhostent	gethostent(3N)	get network host entry
endmntent	getmntent(3)	get file system descriptor file entry
endnetent	getnetent(3N)	get network entry
endnetgrent	getnetgrent(3N)	get network group entry
endprotoent	getprotoent(3N)	get protocol entry
endpwent	getpwent(3)	get password file entry
endservent	getservent(3N)	get service entry
environ	execl(3)	execute a file
erand48	drand48(3)	generate uniformly distributed pseudo-random numbers

erf	erf(3M)	error functions
errno	perror(3)	system error messages
etext	end(3)	last locations in program
ether	ether(3R)	monitor traffic on the Ethernet
ether_aton	ethers(3N)	Ethernet address mapping
ether_hostton	ethers(3N)	Ethernet address mapping
ether_line(3N)	ethers	Ethernet address mapping
ether_ntoa	ethers(3N)	Ethernet address mapping
ether_ntohost	ethers(3N)	Ethernet address mapping
execl	execl(3)	execute a file
execle	execl(3)	execute a file
execlp	execl(3)	execute a file
execv	execl(3)	execute a file
execvp	execl(3)	execute a file
exit	exit(3)	terminate a process after performing cleanup
exp	exp(3M)	exponential function
fabs	floor(3M)	absolute value
fclose	fclose(3S)	close or flush a stream
fcvt	ecvt(3)	output conversion
fdopen	fopen(3S)	open a stream
feof	ferror(3S)	stream status inquiries
ferror	ferror(3S)	stream status inquiries
fflush	fclose(3S)	close or flush a stream
ffs	bstring(3)	bit and byte string operations
fgetc	getc(3S)	get character or integer from stream
fgets	gets(3S)	get a string from a stream
fileno	ferror(3S)	stream status inquiries
finite	ieee(3M)	copysign remainder exponent manipulations
floor	floor(3M)	floor function
fopen	fopen(3S)	open a stream
fprintf	printf(3S)	formatted output conversion
fputc	putc(3S)	put character or word on a stream
fputs	puts(3S)	put a string on a stream
fread	fread(3S)	buffered binary input/output
free	malloc(3)	memory allocator
freopen	fopen(3S)	open a stream
frexp	frexp(3)	split into mantissa and exponent
fscanf	scanf(3S)	formatted input conversion
fseek	fseek(3S)	reposition a stream
ftell	fseek(3S)	reposition a stream
ftime	time(3C)	get date and time
ftok	ftok(3)	standard interprocess communication package
ftw	ftw(3)	walk a file tree
fwrite	fread(3S)	buffered binary input/output
gcvt	ecvt(3)	output conversion
getc	getc(3S)	get character or integer from stream
getchar	getc(3S)	get character or integer from stream
getcwd	getcwd(3)	get pathname of current working directory
getenv	getenv(3)	value for environment name
getfsent	getfsent(3)	get file system descriptor file entry
getfsfile	getfsent(3)	get file system descriptor file entry
getfsspec	getfsent(3)	get file system descriptor file entry
getfstype	getfsent(3)	get file system descriptor file entry

getgrent	getgrent(3)	get group file entry
getgrgid	getgrent(3)	get group file entry
getgrnam	getgrent(3)	get group file entry
gethostbyaddr	gethostent(3N)	get network host entry
gethostbyname	gethostent(3N)	get network host entry
gethostent	gethostent(3N)	get network host entry
getlogin	getlogin(3)	get login name
getmntent	getmntent(3)	get file system descriptor file entry
getnetbyaddr	getnetent(3N)	get network entry
getnetbyname	getnetent(3N)	get network entry
getnetent	getnetent(3N)	get network entry
getnetgrent	getnetgrent(3N)	get network group entry
getopt	getopt(3)	get option letter from argv
getpass	getpass(3)	read a password
getprotobyname	getprotoent(3N)	get protocol entry
getprotobynumber	getprotoent(3N)	get protocol entry
getprotoent	getprotoent(3N)	get protocol entry
getpw	getpw(3)	get name from uid
getpwent	getpwent(3)	get password file entry
getpwnam	getpwent(3)	get password file entry
getpwuid	getpwent(3)	get password file entry
getrpcbyname	getrpcent(3N)	get RPC entry
getrpcbynumber	getrpcent(3N)	get RPC entry
getrpcent	getrpcent(3N)	get RPC entry
getrpcport	getrpcport(3R)	get RPC port number
gets	gets(3S)	get a string from a stream
getservbyname	getservent(3N)	get service entry
getservbyport	getservent(3N)	get service entry
getservent	getservent(3N)	get service entry
getw	getc(3S)	get character or integer from stream
getwd	getwd(3)	get current working directory pathname
gmtime	ctime(3)	convert date and time to ASCII
gsignal	signal(3)	software signals
gtty	stty(3C)	set and get terminal state
hasmntopt	getmntent(3)	get file system descriptor file entry
havedisk	rstat(3R)	get remote host performance data
hcreate	hsearch(3)	manage hash search tables
hdestroy	hsearch(3)	manage hash search tables
hsearch	hsearch(3)	manage hash search tables
htonl	byteorder(3N)	convert values between host and network byte order
htons	byteorder(3N)	convert values between host and network byte order
hypot	hypot(3M)	Euclidean distance
ieee	ieee(3M)	copysign remainder exponent manipulations
index	string(3)	string operations
inet_addr	inet(3N)	Internet address manipulation
inet_lnaof	inet(3N)	Internet address manipulation
inet_makeaddr	inet(3N)	Internet address manipulation
inet_netof	inet(3N)	Internet address manipulation
inet_network	inet(3N)	Internet address manipulation
inet_ntoa	inet(3N)	Internet address manipulation
initgroups	initgroups(3)	initialize group access list
initstate	random(3)	better random number generator
innetgr	getnetgrent(3N)	get network group entry

insque	insque(3)	insert/remove element from a queue
isalnum	ctype(3)	character classification and conversion macros
isalpha	ctype(3)	character classification and conversion macros
isascii	ctype(3)	character classification and conversion macros
isatty	ttyname(3)	find name of a terminal
iscntrl	ctype(3)	character classification and conversion macros
isdigit	ctype(3)	character classification and conversion macros
isgraph	ctype(3)	character classification and conversion macros
isinf	isinf(3)	test for indeterminate floating point values
islower	ctype(3)	character classification and conversion macros
isnan	isinf(3)	test for indeterminate floating point values
isprint	ctype(3)	character classification and conversion macros
ispunct	ctype(3)	character classification and conversion macros
isspace	ctype(3)	character classification and conversion macros
isupper	ctype(3)	character classification and conversion macros
isxdigit	ctype(3)	character classification and conversion macros
j0	j0(3M)	Bessel functions
j1	j0(3M)	Bessel functions
jn	j0(3M)	Bessel functions
jrand48	drand48(3)	generate uniformly distributed pseudo-random numbers
l64a	a64l(3)	convert long to base-64 ASCII
lcong48	drand48(3)	generate uniformly distributed pseudo-random numbers
ldexp	frexp(3)	split into mantissa and exponent
lfind	lsearch(3)	linear search and update
lgamma	lgamma(3M)	log gamma function
localtime	ctime(3)	convert date and time to ASCII
lockf	lockf(3)	advisory record locking on files
log	exp(3M)	exponential functions
log10	exp(3M)	exponential functions
logb	ieee(3M)	copysign remainder exponent manipulations
longjmp	setjmp(3)	non-local goto
lrand48	drand48(3)	generate uniformly distributed pseudo-random numbers
lsearch	lsearch(3)	linear search and update
malloc	malloc(3)	memory allocator
malloc_debug	malloc(3)	memory allocator
malloc_verify	malloc(3)	memory allocator
matherr	matherr(3M)	math library error-handling function
memalign	malloc(3)	memory allocator
memccpy	memory(3)	memory operations
memchr	memory(3)	memory operations
memcmp	memory(3)	memory operations
memcpy	memory(3)	memory operations
memset	memory(3)	memory operations
mkstemp	mktemp(3)	make a unique file name
mktemp	mktemp(3)	make a unique file name
modf	frexp(3)	split into mantissa and exponent
moncontrol	monitor(3)	prepare execution profile
monitor	monitor(3)	prepare execution profile
monstartup	monitor(3)	prepare execution profile
mrand48	drand48(3)	generate uniformly distributed pseudo-random numbers
nice	nice(3C)	set program priority
nlist	nlist(3)	get entries from name list
nrand48	drand48(3)	generate uniformly distributed pseudo-random numbers

ntohl	byteorder(3N)	convert values between host and network byte order
ntohs	byteorder(3N)	convert values between host and network byte order
on_exit	onexit(3)	name termination handler
opendir	directory(3)	directory operations
openlog	syslog(3)	control system log
optarg	getopt(3)	get option letter from argv
optind	getopt(3)	get option letter from argv
pause	pause(3C)	stop until signal
pclose	popen(3S)	initiate I/O to/from a process
perror	perror(3)	system error messages
popen	popen(3S)	initiate I/O to/from a process
pow	exp(3M)	exponential functions
printf	printf(3S)	formatted output conversion
prof	prof(3)	profile within a function
psignal	psignal(3)	system signal messages
putc	putc(3S)	put character or word on a stream
putchar	putc(3S)	put character or word on a stream
putenv	utenv(3)	change or add value to environment
putpwent	putpwent(3)	write password file entry
puts	puts(3S)	put a string on a stream
putw	putc(3S)	put character or word on a stream
qsort	qsort(3)	quicker sort
rand	rand(3C)	random number generator
random	random(3)	better random number generator
rcmd	rcmd(3N)	routines for returning a stream to a remote command
re_comp	regex(3)	regular expression handler
re_exec	regex(3)	regular expression handler
readdir	directory(3)	directory operations
realloc	malloc(3)	memory allocator
regexp	regexp(3)	regular expression compile and match routines
remque	insque(3)	insert/remove element from a queue
rewind	fseek(3S)	reposition a stream
rewinddir	directory(3)	directory operations
rex	rex(3R)	remote execution protocol
rexec	rexec(3N)	return stream to a remote command
rindex	string(3)	string operations
rint	floor(3M)	round to nearest integer
rnusers	rnusers(3R)	return info about users on remote hosts
rquota	rquota(3R)	implement quotas on remote hosts
rresvport	rcmd(3N)	routines for returning a stream to a remote command
rstat	rstat(3R)	get remote host performance data
ruserok	rcmd(3N)	routines for returning a stream to a remote command
rusers	rnusers(3R)	return info about users on remote hosts
rwall	rwall(3R)	write to remote host
scalb	ieee(3M)	copysign remainder exponent manipulations
scandir	scandir(3)	scan a directory
scanf	scanf(3S)	formatted input conversion
seed48	drand48(3)	generate uniformly distributed pseudo-random numbers
seekdir	directory(3)	directory operations
setbuf	setbuf(3S)	assign buffering to a stream
setbuffer	setbuf(3S)	assign buffering to a stream
setegid	setuid(3)	set user and group ID
seteuid	setuid(3)	set user and group ID

setfsent	getfsent(3)	get file system descriptor file entry
setgid	setuid(3)	set user and group ID
setgrent	getgrent(3)	get group file entry
sethostent	gethostent(3N)	get network host entry
setjmp	setjmp(3)	non-local goto
setkey	crypt(3)	DES encryption
setlinebuf	setbuf(3S)	assign buffering to a stream
setlinebuf	setbuf(3S)	assign buffering to a stream
setmntent	getmntent(3)	get file system descriptor file entry
setnetent	getnetent(3N)	get network entry
setnetgrent	getnetgrent(3N)	get network group entry
setprotoent	getprotoent(3N)	get protocol entry
setpwent	getpwent(3)	get password file entry
setrgid	setuid(3)	set user and group ID
setruid	setuid(3)	set user and group ID
setservent	getservent(3N)	get service entry
setstate	random(3)	better random number generator
setuid	setuid(3)	set user and group ID
setvbuf	setbuf(3S)	assign buffering to a stream
siginterrupt	siginterrupt(3)	allow signals to interrupt system calls
signal	signal(3)	simplified software signal facilities
sin	sin(3M)	trigonometric functions
sinh	sinh(3M)	hyperbolic functions
sleep	sleep(3)	suspend execution for interval
spray	spray(3R)	scatter packets to check network
sprintf	printf(3S)	formatted output conversion
sqrt	sqrt(3M)	square root
srand	rand(3C)	random number generator
srand48	drand48(3)	generate uniformly distributed pseudo-random numbers
srandom	random(3)	better random number generator
sscanf	scanf(3S)	formatted input conversion
ssignal	ssignal(3)	software signals
stdio	intro(3S)	standard buffered input/output package
strcat	string(3)	string operations
strcmp	string(3)	string operations
strcpy	string(3)	string operations
strlen	string(3)	string operations
strncat	string(3)	string operations
strncmp	string(3)	string operations
strncpy	string(3)	string operations
strtod	strtod(3)	convert string to double-precision number
strtol	strtol(3)	convert string to integer
stty	stty(3C)	set and get terminal state
swab	swab(3)	swap bytes
sys_errlist	perror(3)	system error messages
sys_nerr	perror(3)	system error messages
sys_siglist	psignal(3)	system signal messages
syslog	syslog(3)	control system log
system	system(3)	issue a shell command
tan	sin(3M)	trigonometric functions
tanh	sinh(3M)	hyperbolic functions
tdelete	tsearch(3)	manage binary search trees
telldir	directory(3)	directory operations

tfind	tsearch(3)	manage binary search trees
time	time(3C)	get date and time
times	times(3C)	get process times
timezone	ctime(3)	convert date and time to ASCII
tmpfile	tmpfile(3S)	create a temporary file
tmpnam	tmpnam(3S)	create a name for a temporary file
toascii	ctype(3)	character classification and conversion macros
tolower	ctype(3)	character classification and conversion macros
toupper	ctype(3)	character classification and conversion macros
tsearch	tsearch(3)	manage binary search trees
ttyname	ttyname(3)	find name of a terminal
ttyslot	ttyname(3)	find name of a terminal
twalk	tsearch(3)	manage binary search trees
ualarm	ualarm(3)	schedule signal after microsecond interval
ulimit	ulimit(3C)	get and set user limits
ungetc	ungetc(3S)	push character back into input stream
usleep	usleep(3S)	suspend execution for micorsecond interval
utime	utime(3C)	set file times
valloc	valloc(3)	aligned memory allocator
values	values(3)	machine-dependent values
varargs	varargs(3)	variable argument list
vfprintf	vprintf(3S)	print formatted output of a varargs argument list
vlimit	vlimit(3C)	control maximum system resource consumption
vprintf	vprintf(3S)	print formatted output of a varargs argument list
vsprintf	vprintf(3S)	print formatted output of a varargs argument list
vtimes	vtimes(3C)	get information about resource utilization
y0	j0(3M)	Bessel functions
y1	j0(3M)	Bessel functions
yn	j0(3M)	Bessel functions
yp_all	ypclnt(3N)	YP client interface routines
yp_bind	ypclnt(3N)	YP client interface routines
yp_first	ypclnt(3N)	YP client interface routines
yp_get_default_domain	ypclnt(3N)	YP client interface routines
yp_master	ypclnt(3N)	YP client interface routines
yp_match	ypclnt(3N)	YP client interface routines
yp_next	ypclnt(3N)	YP client interface routines
yp_order	ypclnt(3N)	YP client interface routines
yp_unbind	ypclnt(3N)	YP client interface routines
ypclnt	ypclnt(3N)	YP client interface routines
yperr_string	ypclnt(3N)	YP client interface routines
yppasswd	yppasswd(3R)	update user YP password
ypprot_err	ypclnt(3N)	YP client interface routines

NAME

directory, opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>

DIR *opendir(filename)
char *filename;

struct direct *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

seekdir(dirp, loc)
DIR *dirp;
long loc;

rewinddir(dirp)
DIR *dirp;

closedir(dirp)
DIR *dirp;
```

DESCRIPTION

opendir opens the directory named by *filename* and associates a *directory stream* with it. *opendir* returns a pointer to be used to identify the *directory stream* in subsequent operations. The pointer NULL is returned if *filename* cannot be accessed or is not a directory, or if it cannot *malloc*(3) enough memory to hold the whole thing.

readdir returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid *seekdir* operation.

telldir returns the current location associated with the named *directory stream.*

seekdir sets the position of the next *readdir* operation on the *directory stream.* The new position reverts to the one associated with the *directory stream* when the *telldir* operation was performed. Values returned by *telldir* are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the *telldir* value may be invalidated due to undetected directory compaction. It is safe to use a previous *telldir* value immediately after a call to *opendir* and before any calls to *readdir*.

Rewinddir resets the position of the named *directory stream* to the beginning of the directory.

closedir closes the named *directory stream* and frees the structure associated with the DIR pointer.

Sample code which searchs a directory for entry ''name'' is:

```
len = strlen(name);
dirp = opendir(".");
for (dp = readdir(dirp); dp != NULL; dp = readdir(dirp))
        if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
                closedir(dirp);
                return FOUND;
        }
closedir(dirp);
return NOT_FOUND;
```

SEE ALSO

open(2), close(2), read(2), lseek(2), getwd(3), dir(5)

NOTES

All UNIX programs that examine directories must be converted to use this package in Sun release 3.0 and beyond. Direct reading of directories is no longer allowed.

BUGS

The new directory format is not obvious.

NAME

frexp, ldexp, modf – floating point analysis and synthesis

SYNOPSIS

double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;
int exp;

double modf(value, iptr)
double value, *iptr;

DESCRIPTION

Frexp returns the significand of a double *value* as a double quantity, x, of magnitude less than 1 and stores an integer n, indirectly through *eptr*, such that $value = x * 2^n$.

The results are not defined when *value* is an IEEE infinity or NaN.

ldexp returns the quantity:

$value * 2^{exp}$.

modf returns the fractional part of *value* and stores the integral part indirectly through *iptr*. Thus the argument *value* and the returned values *modf* and **iptr* satisfy, in the absence of rounding error,

(**iptr* + *modf*) == *value*

and

0 <= *abs*(*modf*) < *abs*(*value*).

The signs of **iptr* and *modf* are the same as the signs of *value*. The results are not defined when *value* is an IEEE infinity or NaN.

Since Sun's definition of *modf* conforms to the System V Interface Definition and the VAX 4.2BSD implementation but differs from the 4.2BSD documentation, results vary from some other Unix implementations whose *modf* conforms to the 4.2BSD documentation but not the VAX 4.2BSD implementation. Therefore avoid *modf* in code intended to be portable.

SEE ALSO

floor(3m)

NAME

ftok – standard interprocess communication package

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(path, id)
char *path;
char id;

DESCRIPTION

All interprocess communication facilities require the user to supply a key to be used by the *msgget*(2), *semget*(2), and *shmget*(2) system calls to obtain interprocess communication identifiers. One suggested method for forming a key is to use the *ftok* subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

ftok returns a key based on *path* and *id* that is usable in subsequent *msgget*, *semget*, and *shmget* system calls. *path* must be the path name of an existing file that is accessible to the process. *id* is a character which uniquely identifies a project. Note that *ftok* will return the same key for linked files when called with the same *id* and that it will return different keys when called with the same file name but different *ids*.

SEE ALSO

intro(2), msgget(2), semget(2), shmget(2)

DIAGNOSTICS

ftok returns (**key_t**) **–1** if *path* does not exist or if it is not accessible to the process.

WARNING

If the file whose *path* is passed to *ftok* is removed when keys still refer to the file, future calls to *ftok* with the same *path* and *id* will return an error. If the same file is recreated, then *ftok* is likely to return a different key than it did the original time it was called.

NAME

monitor, monstartup, moncontrol – prepare execution profile

SYNOPSIS

monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[];

monstartup(lowpc, highpc)
int (*lowpc)(), (*highpc)();

moncontrol(mode)

DESCRIPTION

There are two different forms of monitoring available: An executable program created by:

cc –p . . .

automatically includes calls for the *prof*(1) monitor and includes an initial call to its start-up routine *monstartup* with default parameters; *monitor* need not be called explicitly except to gain fine control over profil buffer allocation. An executable program created by:

cc –pg . . .

automatically includes calls for the *gprof*(1) monitor.

Monstartup is a high level interface to *profil*(2). *Lowpc* and *highpc* specify the address range that is to be sampled; the lowest address sampled is that of *lowpc* and the highest is just below *highpc*. *Monstartup* allocates space using *sbrk*(2) and passes it to *monitor* (see below) to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. Only calls of functions compiled with the profiling option **–p** of *cc*(1) are recorded.

To profile the entire program, it is sufficient to use

```
extern etext();
. . .
monstartup(0x2000, etext);
```

Etext lies just above all the program text, see *end*(3).

To stop execution monitoring and write the results on the file *mon.out*, use

```
monitor(0);
```

then *prof*(1) can be used to examine the results.

Moncontrol is used to selectively control profiling within a program. This works with either *prof*(1) or *gprof*(1) type profiling. When the program starts, profiling begins. To stop the collection of histogram ticks and call counts use *moncontrol*(0); to resume the collection of histogram ticks and call counts use *moncontrol*(1). This allows the cost of particular operations to be measured. Note that an output file will be produced upon program exit irregardless of the state of *moncontrol*.

Monitor is a low level interface to *profil*(2). *Lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* short integers. At most *nfunc* call counts can be kept. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. *Monitor* divides the buffer into space to record the histogram of program counter samples over the range *lowpc* to *highpc*, and space to record call counts of functions compiled with the **–p** option to *cc*(1).

To profile the entire program, it is sufficient to use

```
extern etext();
. . .
monitor(0x2000, etext, buf, bufsize, nfunc);
```

FILES

mon.out

SEE ALSO

cc(1), prof(1), gprof(1), profil(2), sbrk(2)

NAME

utime – set file times

SYNOPSIS

```
#include <sys/types.h>

utime(file, timep)
char *file;
time_t timep[2];
```

DESCRIPTION

The *utime* call uses the 'accessed' and 'updated' times in that order from the *timep* vector to set the corresponding recorded times for *file*.

The caller must be the owner of the file or the super-user. The 'inode-changed' time of the file is set to the current time.

SEE ALSO

utimes(2), stat(2)

NAME

vlimit – control maximum system resource consumption

SYNOPSIS

#include <sys/vlimit.h>

vlimit(resource, value) int resource, value;

DESCRIPTION

This facility is superseded by getrlimit(2).

Limits the consumption by the current process and each process it creates to not individually exceed *value* on the specified *resource*. If *value* is specified as −1, then the current limit is returned and the limit is unchanged. The resources which are currently controllable are:

LIM_NORAISE A pseudo-limit; if set non-zero then the limits may not be raised. Only the super-user may remove the *noraise* restriction.

LIM_CPU the maximum number of cpu-seconds to be used by each process

LIM_FSIZE the largest single file which can be created

LIM_DATA the maximum growth of the data+stack region via *sbrk*(2) beyond the end of the program text

LIM_STACK the maximum size of the automatically-extended stack region

LIM_CORE the size of the largest core dump that will be created.

LIM_MAXRSS a soft limit for the amount of physical memory (in bytes) to be given to the program. If memory is tight, the system will prefer to take memory from processes which are exceeding their declared LIM_MAXRSS.

Because this information is stored in the per-process information this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to *csh*(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way; a *break* call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file i/o operation which would create a file which is too large will cause a signal SIGXFSZ to be generated, this normally terminates the process, but may be caught. When the cpu time limit is exceeded, a signal SIGXCPU is sent to the offending process; to allow it time to process the signal it is given 5 seconds grace by raising the cpu time limit.

SEE ALSO

csh(1)

BUGS

If LIM_NORAISE is set, then no grace should be given when the cpu time limit is exceeded.

There should be *limit* and *unlimit* commands in *sh*(1) as well as in *csh*.

NAME

fopen, freopen, fdopen – open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

DESCRIPTION

fopen opens the file named by *filename* and associates a stream with it. If the open succeeds, *fopen* returns a pointer to be used to identify the stream in subsequent operations.

filename points to a character string that contains the name of the file to be opened.

type is a character string having one of the following values:

"r"	open for reading
"w"	truncate or create for writing
"a"	append: open for writing at end of file, or create for writing
"r+"	open for update (reading and writing)
"w+"	truncate or create for update
"a+"	append; open or create for update at end-of-file

freopen opens the file named by *filename* and associates the stream pointed to by *stream* with it. The *type* argument is used just as in *fopen*. The original stream is closed, regardless of whether the open ultimately succeeds. If the open succeeds, *freopen* returns the original value of *stream*.

freopen is typically used to attach the preopened streams associated with **stdin**, **stdout**, and **stderr** to other files.

fdopen associates a stream with a file descriptor. File descriptors are obtained from calls like *open*, *dup*, *creat*, or *pipe*(2), which open files but do not return streams. Streams are necessary input for many of the Section 3S library routines. The *type* of the stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *fseek*, *rewind*, or an input operation which encounters end-of-file.

SEE ALSO

open(2V), fclose(3S), fseek(3S), fopen(3V)

DIAGNOSTICS

fopen, *freopen*, and *fdopen* return a NULL pointer on failure.

BUGS

In order to support the same number of open files as the system does, *fopen* must allocate additional memory for data structures using *calloc* after 30 files have been opened. This confuses some programs which use their own memory allocators.

NAME

fread, fwrite – buffered binary input/output

SYNOPSIS

#include <stdio.h>

fread(ptr, size, nitems, stream)
FILE *stream;

fwrite(ptr, size, nitems, stream)
FILE *stream;

DESCRIPTION

fread reads, into a block pointed to by *ptr*, *nitems* of data from the named input *stream*, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length *size*. It returns the number of items actually read. *fread* stops appending bytes if an end-of-file or error condition is encountered while reading *stream*, or if *nitems* items have been read. *fread* leaves the file pointer in *stream*, if defined, pointing to the byte following the last byte read if there is one. *fread* does not change the contents of *stream*.

If the standard output is line-buffered, *fread* flushes its output before reading from the standard input. *This is also true for the standard error.*

fwrite appends at most *nitems* of data from the block pointed to by *ptr* to the named output *stream*. It returns the number of items actually written. *fwrite* stops appending when it has appended *nitems* items of data or if an error condition is encountered on *stream*. *fwrite* does not change the contents of the block pointed to by *ptr*.

The argument *size* is typically *sizeof(*ptr)* where the pseudo-function *sizeof* specifies the length of an item pointed to by *ptr*. If *ptr* points to a data type other than *char* it should be cast into a pointer to *char*.

If *size* or *nitems* is non-positive, no characters are read or written and 0 is returned by both *fread* and *fwrite*.

SEE ALSO

read(2V), write(2V), fopen(3S), getc(3S), putc(3S), gets(3S), puts(3S), printf(3S), scanf(3S), fread(3V)

DIAGNOSTICS

fread and *fwrite* return 0 upon end of file or error.

NAME

puts, fputs – put a string on a stream

SYNOPSIS

```
#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

puts writes the null-terminated string pointed to by *s*, followed by a newline character, to the standard output stream **stdout**.

fputs writes the null-terminated string pointed to by *s* to the named output *stream*.

Neither function writes the terminal null character.

DIAGNOSTICS

Both routines return **EOF** on error. This will happen if the routines try to write on a file that has not been opened for writing.

SEE ALSO

fopen(3S), putc(3S), printf(3S), ferror(3S), fread(3S)

NOTES

puts appends a newline while *fputs* does not.

NAME

scanf, fscanf, sscanf – formatted input conversion

SYNOPSIS

```
#include <stdio.h>

scanf(format [, pointer ] ... )
char *format;

fscanf(stream, format [, pointer ] ... )
FILE *stream;
char *format;

sscanf(s, format [, pointer ] ... )
char *s, *format;
```

DESCRIPTION

scanf reads from the standard input stream **stdin**. *fscanf* reads from the named input *stream*. *sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format*, described below, and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks, tabs, or new-lines) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not %), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, an optional **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except "[" and "c", white space leading an input field is ignored.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion characters are legal:

- **%** a single % is expected in the input at this point; no assignment is done.
- **d** a decimal integer is expected; the corresponding argument should be an integer pointer.
- **u** an unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.
- **o** an octal integer is expected; the corresponding argument should be a integer pointer.
- **x** a hexadecimal integer is expected; the corresponding argument should be an integer pointer.
- **e,f,g** a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an **E** or **e** followed by an optional +, –, or space, followed by an integer.
- **s** a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a white space character.
- **c** a character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use **%1s**. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

[indicates string data; the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first–last*, thus [0123456789] may be expressed [0–9]. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating \0, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters **d**, **u**, **o**, and **x** may be capitalized or preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list The **l** or **h** modifier is ignored for other conversion characters.

scanf conversion terminates at **EOF**, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string.

If the input ends before the first conflict or conversion, **EOF** is returned. If the input ends after the first conflict or conversion, the number of successfully matched items is returned.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to *n* the value **3**, to *i* the value **25**, to *x* the value **5.432**, and *name* will contain **thompson\0**. Or:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0-9]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

will assign **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name*. The next call to *getchar* (see *getc*(3S)) will return **a**.

SEE ALSO

getc(3S), printf(3S) strtod(3), strtol(3), scanf(3V)

DIAGNOSTICS

These functions return **EOF** on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

scanf cannot read the strings which *printf*(3S) generates for IEEE indeterminate floating point values.

scanf provides no way to convert a number in any arbitrary base (decimal, hex or octal) based on the traditional C conventions (leading 0 or 0x).

NAME

setbuf, setbuffer, setlinebuf, setvbuf – assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;

setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

setlinebuf(stream)
FILE *stream;

int setvbuf (stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is encountered or input is read from stdin. *fflush* (see *fclose*(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from *malloc*(3) upon the first *getc* or *putc*(3S) on the file. If the standard stream **stdout** refers to a terminal it is line buffered. If the standard stream **stderr** refers to a terminal it is line buffered.

setbuf can be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered. A manifest constant **BUFSIZ**, defined in the **<stdio.h>** header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

setbuffer, an alternate form of *setbuf*, can be used after a stream has been opened but before it is read or written. It causes the character array *buf* whose size is determined by the *size* argument to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered.

setvbuf can be used after a stream has been opened but before it is read or written. *type* determines how *stream* will be buffered. Legal values for *type* (defined in **<stdio.h>**) are:

_IOFBF	causes input/output to be fully buffered.
_IOLBF	causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested.
_IONBF	causes input/output to be completely unbuffered. If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *Size* specifies the size of the buffer to be used.

setlinebuf is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike *setbuf*, *setbuffer*, and *setvbuf*, it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using *freopen* (see *fopen*(3S)). A file can be changed from block buffered or line buffered to unbuffered by using *freopen* followed by *setbuf* with a buffer argument of NULL.

NAME

scanf, fscanf, sscanf – formatted input conversion

SYNOPSIS

```
#include <stdio.h>

scanf(format [, pointer ] ... )
char *format;

fscanf(stream, format [, pointer ] ... )
FILE *stream;
char *format;

sscanf(s, format [, pointer ] ... )
char *s, *format;
```

DESCRIPTION

scanf reads from the standard input stream **stdin**. *fscanf* reads from the named input *stream*. *sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format*, described below, and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not %), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, an optional **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except "[" and "c", white space leading an input field is ignored.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion characters are legal:

- **%** a single % is expected in the input at this point; no assignment is done.
- **d** a decimal integer is expected; the corresponding argument should be an integer pointer.
- **u** an unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.
- **o** an octal integer is expected; the corresponding argument should be a integer pointer.
- **x** a hexadecimal integer is expected; the corresponding argument should be an integer pointer.
- **e,f,g** a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an **E** or **e** followed by an optional +, –, or space, followed by an integer.
- **s** a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \0, which will be added automatically. The input field is terminated by a white space character.
- **c** a character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use %1s. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

[indicates string data; the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first–last*, thus [0123456789] may be expressed [0–9]. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating \0, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters **d**, **u**, **o**, and **x** may be capitalized or preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list The **l** or **h** modifier is ignored for other conversion characters.

scanf conversion terminates at **EOF**, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string.

If the input ends before the first conflict or conversion, **EOF** is returned. If the input ends after the first conflict or conversion, the number of successfully matched items is returned.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E–1 thompson
```

will assign to *n* the value **3**, to *i* the value **25**, to *x* the value **5.432**, and *name* will contain **thompson\0**. Or:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0–9]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

will assign **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name*. The next call to *getchar* (see *getc*(3S)) will return **a**.

SEE ALSO

getc(3S), printf(3V) strtod(3), strtol(3), scanf(3S)

DIAGNOSTICS

These functions return **EOF** on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

scanf cannot read the strings which *printf*(3V) generates for IEEE indeterminate floating point values.

scanf provides no way to convert a number in any arbitrary base (decimal, hex or octal) based on the traditional C conventions (leading 0 or 0x).

NAME

setbuf, setbuffer, setlinebuf, setvbuf – assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;

setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

setlinebuf(stream)
FILE *stream;

int setvbuf (stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is encountered or input is read from stdin. *fflush* (see *fclose*(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from *malloc*(3) upon the first *getc* or *putc*(3S) on the file.

By default, output to a terminal is line buffered and all other input/output is fully buffered.

setbuf can be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered. A manifest constant **BUFSIZ**, defined in the **<stdio.h>** header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

setbuffer, an alternate form of *setbuf*, can be used after a stream has been opened but before it is read or written. It causes the character array *buf* whose size is determined by the *size* argument to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered.

setvbuf can be used after a stream has been opened but before it is read or written. *type* determines how *stream* will be buffered. Legal values for *type* (defined in **<stdio.h>**) are:

_IOFBF	causes input/output to be fully buffered.
_IOLBF	causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested.
_IONBF	causes input/output to be completely unbuffered. If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *Size* specifies the size of the buffer to be used.

setlinebuf is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike *setbuf*, *setbuffer*, and *setvbuf*, it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using *freopen* (see *fopen*(3S)). A file can be changed from block buffered or line buffered to unbuffered by using *freopen* followed by *setbuf* with a buffer argument of NULL.

NAME

intro – introduction to other libraries

DESCRIPTION

This section contains manual pages describing other libraries, which are available only from C. The list below includes libraries which provide device independent plotting functions, terminal independent screen management routines for two dimensional non-bitmap display terminals, and functions for managing data bases with inverted indexes. All functions are located in separate libraries indicated in each manual entry.

FILES

/usr/lib/libcurses.a	screen management routines (see *curses*(3X))
/usr/lib/libdbm.a	data base management routines (see *dbm*(3X))
/usr/lib/libmp.a	multiple precision math library (see *mp*(3X))
/usr/lib/libplot.a	plot routines (see *plot*(3X))
/usr/lib/lib300.a	"
/usr/lib/lib300s.a	"
/usr/lib/lib450.a	"
/usr/lib/lib4014.a	"
/usr/lib/libtermcap.a	terminal handling routines (see *termcap*(3X))
/usr/lib/libtermcap_p.a	
/usr/lib/libtermlib.a	(link to /usr/lib/libtermcap.a)
/usr/lib/libtermlib_p.a	(link to /usr/lib/libtermcap_p.a)

NAME

curses – screen functions with "optimal" cursor motion

SYNOPSIS

cc [*flags*] *files* **–lcurses –ltermcap** [*libraries*]

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the *refresh()* tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine *initscr()* must be called before any of the other routines that deal with windows and screens are used. The routine *endwin()* should be called before exiting.

SEE ALSO

ioctl(2), getenv(3), tty(4), termcap(5)
Programmer's Reference Manual for Curses

addch(ch)	add a character to *stdscr*
addstr(str)	add a string to *stdscr*
box(win,vert,hor)	draw a box around a window
cbreak()	set cbreak mode
clear()	clear *stdscr*
clearok(scr,boolf)	set clear flag for *scr*
clrtobot()	clear to bottom on *stdscr*
clrtoeol()	clear to end of line on *stdscr*
delch()	delete a character
deleteln()	delete a line
delwin(win)	delete *win*
echo()	set echo mode
endwin()	end window modes
erase()	erase *stdscr*
flusok(win,boolf)	set flush-on-refresh flag for *win*
getch()	get a char through *stdscr*
getcap(name)	get terminal capability *name*
getstr(str)	get a string through *stdscr*
gettmode()	get tty modes
getyx(win,y,x)	get (y,x) co-ordinates
inch()	get char at current (y,x) co-ordinates
initscr()	initialize screens
insch(c)	insert a char
insertln()	insert a line
leaveok(win,boolf)	set leave flag for *win*
longname(termbuf,name)	get long name from *termbuf*
move(y,x)	move to (y,x) on *stdscr*
mvcur(lasty,lastx,newy,newx)	actually move cursor
newwin(lines,cols,begin_y,begin_x)	create a new window
nl()	set newline mapping
nocbreak()	unset cbreak mode
noecho()	unset echo mode
nonl()	unset newline mapping
noraw()	unset raw mode
overlay(win1,win2)	overlay win1 on win2
overwrite(win1,win2)	overwrite win1 on top of win2
printw(fmt,arg1,arg2,...)	printf on *stdscr*
raw()	set raw mode
refresh()	make current screen look like *stdscr*

NAME

bwone – Sun-1 black and white frame buffer

SYNOPSIS — SUN-2

device bwone0 at mbmem ? csr 0xc0000 priority 3

DESCRIPTION

The *bwone* interface provides access to Sun-1 black and white graphics controller boards. It supports the ioctls described in *fbio*(4S).

FILES

/dev/bwone[0-9]

SEE ALSO

mmap(2), fb(4S), fbio(4S)

BUGS

Use of vertical-retrace interrupts is not supported.

The video state returned by the FBIOGVIDEO ioctl may be incorrect. It is not possible for the driver to determine the state of the hardware video enable bit, so it reports the last state stored by the FBIOSVIDEO ioctl. User processes which map the frame buffer can directly enable or disable the video, unknown to the driver.

NAME

bwtwo – Sun-3/Sun-2 black and white frame buffer

SYNOPSIS — SUN-3

device bwtwo0 at obmem 1 csr 0xff000000 priority 4
device bwtwo0 at obmem 2 csr 0x100000 priority 4
device bwtwo0 at obmem 3 csr 0xff000000 priority 4
device bwtwo0 at obmem 4 csr 0xff000000

The first synopsis line given above should be used to generate a kernel for a Sun-3/75 or Sun-3/160; the second, for a Sun-3/50; the third, for a Sun-3/260; and the fourth, for a Sun-3/110.

SYNOPSIS — SUN-2

device bwtwo0 at obmem 1 csr 0x700000 priority 4
device bwtwo0 at obio 2 csr 0x0 priority 4

The first synopsis line given above should be used to generate a kernel for a Sun-2/120 or Sun-2/170; the second, for a Sun-2/50 or Sun-2/160.

DESCRIPTION

The *bwtwo* interface provides access to Sun monochrome memory frame buffers. It supports the ioctls described in *fbio*(4S).

If **flags 0x1** is specified, frame buffer write operations are buffered through regular high-speed RAM. This "copy memory" mode of operation speeds frame buffer accesses, but consumes an extra 128K bytes of memory. Only the Sun-3/50, Sun-3/75, and Sun-3/160 support copy memory; on other systems a warning message will be printed and the flag will be ignored.

Reading or writing to the frame buffer is not allowed — you must use the *mmap*(2) system call to map the board into your address space.

FILES

/dev/bwtwo[0-9]

SEE ALSO

mmap(2), fb(4S), fbio(4S), cgfour(4S)

BUGS

Use of vertical-retrace interrupts is not supported.

NAME

cgfour – Sun-3 color memory frame buffer

SYNOPSIS — SUN-3

device cgfour0 at obmem 4 csr

DESCRIPTION

The *cgfour* is a color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented on the Sun-3/110 and Sun-3/160. It provides the standard frame buffer interface as defined in *fbio*(4S).

In addition to the ioctls described under *fbio*(4s), the *cgfour* interface responds to two *cgfour*-specific colormap ioctls, FBIOPUTCMAP and FBIOGETCMAP. FBIOPUTCMAP returns no information other than success/failure via the ioctl return value. FBIOGETCMAP returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in *<sun/fbio.h>* as:

```
struct fbcmap {
        int             index;          /* first element (0 origin) */
        int             count;          /* number of elements */
        unsigned char   *red;           /* red color map elements */
        unsigned char   *green;         /* green color map elements */
        unsigned char   *blue;          /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

FILES

/dev/cgfour0

SEE ALSO

mmap(2), fbio(4S)

NAME

cgone – Sun-1 color graphics interface

SYNOPSIS — SUN-2

device cgone0 at mbmem ? csr 0xec000 priority 3

DESCRIPTION

The *cgone* interface provides access to the Sun-1 color graphics controller board, which is normally supplied with a 13" or 19" RS170 color monitor. It provides the standard frame buffer interface as defined in *fbio*(4S).

It supports the FBIOGPIXRECT ioctl which allows SunWindows to be run on it; see *fbio*(4S)

The hardware consumes 16 kilobytes of Multibus memory space. The board starts at standard addresses 0xE8000 or 0xEC000. The board must be configured for interrupt level 3.

FILES

/dev/cgone[0-9]

SEE ALSO

mmap(2), fbio(4S)

BUGS

Use of color board vertical-retrace interrupts is not supported.

NAME

fbio – general properties of frame buffers

DESCRIPTION

All of the Sun frame buffers support the same general interface. Each responds to a FBIOGTYPE ioctl which returns information in a structure defined in *<sun/fbio.h>*:

```
struct  fbtype {
        int     fb_type;        /* as defined below */
        int     fb_height;      /* in pixels */
        int     fb_width;       /* in pixels */
        int     fb_depth;       /* bits per pixel */
        int     fb_cmsize;      /* size of color map (entries) */
        int     fb_size;        /* total size in bytes */
};

#define FBTYPE_SUN1BW       0
#define FBTYPE_SUN1COLOR    1
#define FBTYPE_SUN2BW       2
#define FBTYPE_SUN2COLOR    3
#define FBTYPE_SUN2GP       4
#define FBTYPE_SUN4COLOR    8
```

Each device has an FBTYPE which is used by higher-level software to determine how to perform raster-op and other functions. Each device is used by opening it, doing an FBIOGTYPE ioctl to see which frame buffer type is present, and thereby selecting the appropriate device-management routines.

Full-fledged frame buffers (that is, those that run SunWindows) implement an FBIOGPIXRECT ioctl, which returns a pixrect. This call is made only from inside the kernel. The returned pixrect is used by *win*(4S) for cursor tracking and colormap loading.

FBIOSVIDEO and FBIOGVIDEO are general-purpose ioctls for controlling possible video features of frame buffers. They are defined in *<sun/fbio.h>* . These ioctls either set or return the value of a flags integer. At this point, only the FBVIDEO_ON option is available, controlled by FBIOSVIDEO. FBIOGVIDEO returns the current video state.

The FBIOSATTR and FBIOGATTR ioctls allow access to special features of newer frame buffers. They use the following structures as defined in *<sun/fbio.h>*:

```
#define  FB_ATTR_NDEVSPECIFIC     8       /* no. of device specific values */
#define  FB_ATTR_NEMUTYPES        4       /* no. of emulation types */

struct fbsattr {
        int     flags;                   /* misc flags */
#define  FB_ATTR_AUTOINIT         1       /* emulation auto init flag */
#define  FB_ATTR_DEVSPECIFIC      2       /* dev. specific stuff valid flag */
        int     emu_type;                /* emulation type (-1 if unused) */
        int     dev_specific[FB_ATTR_NDEVSPECIFIC];/* catchall */
};

struct fbgattr {
        int     real_type;               /* real device type */
        int     owner;                   /* PID of owner, 0 if myself */
        struct fbtype fbtype;            /* fbtype info for real device */
        struct fbsattr sattr;            /* see above */
        int     emu_types[FB_ATTR_NEMUTYPES];    /* possible emulations */
                                         /* (-1 if unused) */
};
```

SEE ALSO

mmap(2), bwone(4S), bwtwo(4S), cgone(4S), cgtwo(4S), cgfour(4S), gpone(4S), fb(4S), win(4S)

BUGS

FBIOSATTR and FBIOGATTR are only supported by the *cgfour*(4S) frame buffer.

The FBVIDEO_ON flag my be incorrect for Sun-1 black and white frame buffers; see *bwone*(4S).

NAME

gpone – Sun-3/Sun-2 graphics processor

SYNOPSIS — SUN-3

device gpone0 at vme24d16 ? csr

SYNOPSIS — SUN-2

device gpone0 at vme24 ? csr

DESCRIPTION

The *gpone* interface provides access to the optional Graphics Processor Board (GP).

The hardware consumes 64 kilobytes of VME bus address space. The GP board starts at standard address 0x210000 and must be configured for interrupt level 3.

GP IOCTL

The graphics processor responds to a number of ioctl calls as described here. One of the calls uses a **gp1fbinfo** structure that looks like this:

```
struct   gp1fbinfo {
             int          fb_vmeaddr;    /* physical color board address */
             int          fb_hwwidth;    /* fb board width */
             int          fb_hwheight;   /* fb board height */
             int          addrdelta;     /* phys addr diff between fb and gp */
             caddr_t      fb_ropaddr;    /* cg2 va thru kernelmap */
             int          fbunit;        /* fb unit to use  for a,b,c,d */
};
```

The ioctl call looks like this:

ioctl(file, request, argp)
int file, request;

argp is defined differently for each GP ioctl request and is specified in the descriptions below.

The following ioctl commands provide for transferring data between the graphics processor and color boards and processes.

GP1IO_PUT_INFO
: Passes information about the frame buffer into driver. **argp** points to a **struct gp1fbinfo** which is passed to the driver.

GP1IO_GET_STATIC_BLOCK
: Hands out a static block from the GP. **argp** points to an **int** which is returned from the driver.

GP1IO_FREE_STATIC_BLOCK
: Frees a static block from the GP. **argp** points to an **int** which is passed to the driver.

GP1IO_GET_GBUFFER_STATE
: Checks to see if there is a buffer present on the GP. **argp** points to an **int** which is returned from the driver.

GP1IO_CHK_GP
: Restarts the GP if necessary. **argp** points to an **int** which is passed to the driver.

GP1IO_GET_RESTART_COUNT
: Returns the number of restarts of a GP since power on. Needed to differentiate SIGXCPU calls in user processes. **argp** points to an **int** which is returned from the driver.

GP1IO_REDIRECT_DEVFB
: Configures */dev/fb* to talk to a graphics processor device. **argp** points to an **int** which is passed to the driver.

GP1IO_GET_REQDEV
: Returns the requested minor device. **argp** points to a **dev_t** which is returned from the driver.

GP1IO_GET_TRUMINORDEV
Returns the true minor device. **argp** points to a **char** which is returned from the driver.

The graphics processor driver also responds to the FBIOGTYPE, ioctl which a program can use to inquire as to the characteristics of the display device, the FBIOGINFO, ioctl for passing generic information, and the FBIOGPIXRECT ioctl so that SunWindows can run on it. See *fbio*(4S).

FILES

/dev/gpone[0-3][abcd]
/usr/include/sun/gpio.h
/usr/include/pixrect/{gp1cmds.h,gp1reg.h,gp1var.h}

SEE ALSO

fbio(4S), mmap(2), gpconfig(8)
Software Interface Manual for the Sun Graphics Processor (Part Number: 800-1571-01)

DIAGNOSTICS

The Graphics Processor has been restarted. You may see display garbage as a result.

NAME

mcp, alm – Sun MCP Multiprotocol Communications Processor/ALM-2 Asynchronous Line Multiplexer

CONFIG — SUN-3

MCP and ALM-2

device mcp0 at vme32d32 ? csr 0x1000000 flags 0x1ffff priority 4 vector mcpintr 0x8b
device mcp1 at vme32d32 ? csr 0x1010000 flags 0x1ffff priority 4 vector mcpintr 0x8a
device mcp2 at vme32d32 ? csr 0x1020000 flags 0x1ffff priority 4 vector mcpintr 0x89
device mcp3 at vme32d32 ? csr 0x1030000 flags 0x1ffff priority 4 vector mcpintr 0x88

ALM-2

pseudo-device mcpa64

DESCRIPTION

MCP

The Sun MCP Multiprotocol Communications Processor board supports up to four synchronous serial lines. For more information about the MCP, consult the SunLink™ Multiple Communication Protocol product manuals.

ALM-2

The Sun ALM-2 Asynchronous Line Multiplexer provides 16 asynchronous serial communication lines with modem control and one Centronics-compatible parallel printer port.

Each serial line behaves as described in *tty*(4). Input and output for each line may independently be set to run at any of 16 speeds; see *tty*(4) for the encoding.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of mcp0 would cause line ttyh2 to be treated in this way.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 0–63 correspond directly to the normal tty lines. They are named *ttyXY* where *X* represents the physical board as one of the characters *h, i, j,* or *k,* and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is */dev/ttyh0,* and the sixteenth line on the third board is */dev/ttyjf.*)

Minor device numbers in the range 128–191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128) and are (conventionally) named *cua**. The *cua* lines are special in that they can always be opened with or without a carrier on the line. Once a *cua* line is opened, the corresponding tty line can not be opened until the *cua* line is closed. Also, if the *tty* line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding *cua* line can not be opened.

This allows a modem to be attached to */dev/ttyh0* (usually renamed to */dev/ttyd0*) and used for dialin (by enabling the line for login in */etc/ttys*) and also used for dialout (by *tip*(1C) or *uucp*(1C)) as */dev/cua0* when no one is logged in on the line. Note that the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

PRINTER PORT

The printer port is Centronics-compatible and is suitable for most common parallel printers. Devices attached to this interface are normally handled by the line printer spooling system, and should not be accessed directly by the user.

Minor device numbers in the range 64–67 access the printer port, and the recommended naming is */dev/mcpp[0-3].*

Device Status

Normally, the interface only reports the status of the device when attempting an *open*(2) call.

Errors

Opening the device */dev/mcpp** may yield one of two errors:

ENXIO indicates that the device is already in use

EIO indicates that the device is offline or out of paper.

Bit 17 of the configuration **flags** may be specified to say that the interface should ignore Centronics SLCT– and RDY/PE– when attempting to open the device, but this is normally useful only for configuration and troubleshooting: if the SLCT– and RDY lines are not asserted during an actual data transfer (as with a *write*(2) call), no data is transferred.

Optionally, the interface may be set to print a message to the console when either the paper-out or offline conditions obtain. To set this mode, you should include **<sundev/mcpcmd.h>** and use the *ioctl*(2) call:

```
unsigned char mode;
ioctl(fd, MCPIOSPR, &mode);
```

The bits in are defined as follows:

MCPRINTSLCT	0x20	notify on console if device goes offline
MCPRINTPE	0x10	notify on console if device runs out of paper
MCPRDIAG	0x04	set self-test mode (not ordinarily useful)

The ioctl call:

```
ioctl(fd, MCPIOGPR, &mode);
```

returns in *mode* the current status of the parallel port, defined as follows:

MCPRIGNSLCT	0x02	set if interface ignoring SLCT– on open
MCPRDIAG	0x04	set if in self-test loopback mode
MCPRVMEINT	0x08	set if VME bus interrupts enabled
MCPRINTPE	0x10	set if paper-out notification enabled
MCPRINTSLCT	0x20	set if offline notification enabled
MCPRPE	0x40	set if device ready, cleared if device out of paper
MCPRSLCT	0x80	set if device online (Centronics SLCT asserted)

FILES

/dev/mcpp[0-3]	parallel printer port
/dev/tty[h-k][0-9a-f]	hardwired tty lines
/dev/ttyd[0-9a-f]	dialin tty lines
/dev/cua[0-9a-f]	dialout tty lines

SEE ALSO

tty(4)

DIAGNOSTICS

Most of these diagnostics "should never happen;" their occurrence usually indicates problems elsewhere in the system as well.

mcp*n*: silo overflow.
: More than *k* characters (*k* very large) have been received by the *mcp* hardware without being read by the software.

port *n* supports RS449 interface
: Probably an incorrect jumper configuration. Consult the hardware manual.

mcp port *n* receive buffer error
: The *mcp* encountered an error concerning the synchronous receive buffer.

Printer on mcpp*n* is out of paper

Printer on mcpp*n* paper ok

Printer on mcpp*n* is offline

Printer on mcpp*n* online

Assorted printer diagnostics, if enabled as discussed above.

NAME

xy – Disk driver for Xylogics SMD Disk Controllers

SYNOPSIS — SUN-3

controller xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk xy0 at xyc0 drive 0
disk xy1 at xyc0 drive 1
disk xy2 at xyc1 drive 0
disk xy3 at xyc1 drive 1

The two **controller** lines given in the synopsis sections above specify the first and second Xylogics 450 SMD disk controller in a Sun system.

SYNOPSIS — SUN-2

controller xyc0 at vme16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller xyc1 at vme16 ? csr 0xee48 priority 2 vector xyintr 0x49
controller xyc0 at mbio ? csr 0xee40 priority 2
controller xyc1 at mbio ? csr 0xee48 priority 2
disk xy0 at xyc0 drive 0
disk xy1 at xyc0 drive 1
disk xy2 at xyc1 drive 0
disk xy3 at xyc1 drive 1

The first two **controller** lines specify the first and second Xylogics 450 SMD disk controllers in a Sun-2/160 VMEbus based system. The third and fourth **controller** lines specify the first and second Xylogics 450 SMD disk controllers in a Sun-2/120 or a Sun-2/170 Multibus based system.

DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, and so on. The standard device names begin with 'xy' followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in only one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r'.

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise *seek*(2) calls should specify a multiple of 512 bytes.

If **flags 0x1** is specified, the overlapped seeks feature for that drive is turned off. Note that to be effective, the flag must be set on all drives for a specific controller. This action is necessary for controllers with older firmware, which have bugs preventing overlapped seeks from working properly.

DISK SUPPORT

This driver handles all SMD drives by reading a label from sector 0 of the drive which describes the disk geometry and partitioning.

The **xy?a** partition is normally used for the root file system on a disk, the xy?b partition as a paging area, and the **xy?c** partition for pack-pack copying (it normally maps the entire disk). The rest of the disk is normally the **xy?g** partition.

FILES

/dev/xy[0-7][a-h]	block files
/dev/rxy[0-7][a-h]	raw files

SEE ALSO

dkio(4S)

DIAGNOSTICS

xyc*n*: self test error

Self test error in controller, see the Maintenance and Reference Manual.

xyc*n*: WARNING: *n* bit addresses

The controller is strapped incorrectly. Sun systems use 20-bit addresses for Multibus based systems and 24-bit addresses for VMEbus based systems.

xy*n*: unable to read bad sector info

The bad sector forwarding information for the disk could not be read.

xy*n* and xy*n* are of same type (*n*) with different geometries.

The 450 does not support mixing the drive types found on these units on a single controller.

xy*n*: initialization failed

The drive could not be successfully initialized.

xy*n*: unable to read label

The drive geometry/partition table information could not be read.

xy*n*: Corrupt label

The geometry/partition label checksum was incorrect.

xy*n*: offline

A drive ready status is no longer detected, so the unit has been logically removed from the system. If the drive ready status is restored, the unit will automatically come back online the next time it is accessed.

xy*nc*: *cmd how* (*msg*) blk #*n* abs blk #*n*

A command such as read or write encountered an error condition (how): either it *failed*, the controller was *reset*, the unit was *restored*, or an operation was *retry*'ed. The *msg* is derived from the error number given by the controller, indicating a condition such as "drive not ready", "sector not found" or "disk write protected". The *blk #* is the sector in error relative to the beginning of the partition involved. The *abs blk #* is the absolute block number of the sector in error. Some fields of the error message may be missing since the information is not always available.

BUGS

In raw I/O *read*(2) and *write*(2) truncate file offsets to 512-byte block boundaries, and *write*(2) scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read*(2), *write*(2) and *lseek*(2) should always deal in 512-byte multiples.

Older revisions of the firmware do not properly support overlapped seeks. This will only affect systems with multiple disks on a single controller. If a large number of "zero sector count" errors appear, you should use the **flags** field to disable overlapped seeks.

NAME

acct – execution accounting file

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

The *acct*(2) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

```
/*      @(#)acct.h 1.1 86/07/07 SMI; from UCB 6.1 83/07/29*/

/*
 * Accounting structures;
 * these use a comp_t type which is a 3 bits base 8
 * exponent, 13 bit fraction "floating point" number.
 */
typedef u_short comp_t;

struct  acct
{
        char        ac_comm[10];  /* Accounting command name */
        comp_t      ac_utime;     /* Accounting user time */
        comp_t      ac_stime;     /* Accounting system time */
        comp_t      ac_etime;     /* Accounting elapsed time */
        time_t      ac_btime;     /* Beginning time */
        short       ac_uid;       /* Accounting user ID */
        short       ac_gid;       /* Accounting group ID */
        short       ac_mem;       /* average memory usage */
        comp_t      ac_io;        /* number of disk IO blocks */
        dev_t       ac_tty;       /* control typewriter */
        char        ac_flag;      /* Accounting flag */
};

#define AFORK    0001           /* has executed fork, but no exec */
#define ASU      0002           /* used super-user privileges */
#define ACOMPAT  0004           /* used compatibility mode */
#define ACORE    0010           /* dumped core */
#define AXSIG    0020           /* killed by a signal */

#ifdef KERNEL
#ifdef SYSACCT
struct  acct        acctbuf;
struct  vnode       *acctp;
#else
#define acct()
#endif
#endif
```

If the process does an *execve*(2), the first 10 characters of the filename appear in *ac_comm*. The accounting flag contains bits indicating whether *execve*(2) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

acct(2), execve(2), sa(8)

NAME

aliases, addresses, forward – addresses and aliases for sendmail(8)

SYNOPSIS

/etc/passwd
/usr/lib/aliases
/usr/lib/aliases.dir
/usr/lib/aliases.pag
~/.forward

DESCRIPTION

These files contain mail addresses or aliases, recognized by *sendmail*(8), for the local host:

/etc/passwd Mail addresses (usernames) of local users.

/usr/lib/aliases Aliases for the local host, in ASCII format. This file can be edited to add, update, or delete local mail aliases.

/usr/lib/aliases.{dir,pag}
The aliasing information from */usr/lib/aliases*, in binary, *dbm*(3X) format for use by *sendmail*(8). The program *newaliases*(8), which is invoked automatically by *sendmail*(8), maintains these files.

~/.forward Addresses to which a user's mail is forwarded (see *Automatic Forwarding*, below).

In addition, the Yellow Pages aliases map *mail.aliases* contains addresses and aliases available for use across the network.

ADDRESSES

As distributed, *sendmail*(8) supports the following types of addresses:

- Local usernames. These are listed in the local host's */etc/passwd* file.
- Local filenames. When mailed to an absolute pathname, a message can be appeneded to a file.
- Commands. If the first character of the address is a vertical bar, (|), *sendmail (8)* pipes the message to the standard input of the command the bar precedes.
- DARPA-standard mail addresses of the form:

  name@domain

  If *domain* does not contain any dots (.), then it is interpreted as the name of a host in the current domain. Otherwise, the message is passed to a *mailhost* that determines how to get to the specified domain. Domains are divided into subdomains separated by dots, with the top-level domain on the right. Top-level domains include:

  .COM Commerical organizations.
  .EDU Educational organizations.
  .GOV Government organizations.
  .MIL Military organizations.

  For example, the full address of John Smith could be:

  js@jsmachine.Podunk-U.EDU

  if he uses the machine named "jsmachine" at Podunk University.

- *uucp*(1C) addresses of the form:

  ... [*host*!]*host*!*username*

  These are sometimes mistakenly referred to as "Usenet" addresses. *uucp*(1C) provides links to numerous sites throughout the world for the remote copying of files.

Other site-specific forms of addressing can be added by customizing the *sendmail* configuration file. See the *sendmail*(8), and *Sendmail Installation and Operation* in *System Administration for the Sun Workstation* for details. Standard addresses are recommended.

ALIASES

Local Aliases

/usr/lib/aliases is formatted as a series of lines of the form

name: *address*[, *address*]

name is the name of the alias or alias group, and *address* is the address of a recipient in the group. Aliases can be nested. That is, an *address* can be the name of another alias group. Lines beginning with white space are treated as continuation lines for the preceding alias. Lines beginning with # are comments.

Special Aliases

An alias of the form:

owner–*aliasname*: *address*

directs error-messages resulting from mail to *alias-name* to *address*, instead of back to the person who sent the message.

An alias of the form:

aliasname: **:include:***pathname*

with colons as shown, adds the recipients listed in the file *pathname* to the *aliasname* alias. This allows a private list to be maintained separately from the aliases file.

YP Domain Aliases

Normally, the aliases file on the master YP server is used for the *mail.aliases* YP map, which can be made available to every YP client. Thus, the */usr/lib/aliases** files on the various hosts in a network will one day be obsolete. Domain-wide aliases should ultimately be resolved into usernames on specific hosts. For example, if the following were in the domain-wide alias file:

jsmith:js@jsmachine

then any YP client could just mail to "jsmith" and not have to remember the machine and user name for John Smith. If a YP alias does not resolve to an address with a specific host, then the name of the YP domain is used. There should be an alias of the domain name for a host in this case. For example, the alias:

jsmith:root

sends mail on a YP client to "root@podunk-u" if the name of the YP domain is "podunk-u".

Automatic Forwarding

When an alias (or address) is resolved to the name of a user on the local host, *sendmail* checks for a *.forward* file, owned by the intended recipient, in that user's home directory, and with universal read access. This file can contain one or more addresses or aliases as described above, each of which is sent a copy of the user's mail.

Care must be taken to avoid creating addressing loops in the *.forward* file. When forwarding mail between machines, be sure that the destination machine does not return the mail to the sender through the operation of any YP aliases. Otherwise, copies of the message may "bounce." Usually, the solution is to change the YP alias to direct mail to the proper destination.

A backslash before a username inhibits further aliasing. For instance, to invoke the *vacation*(1) program, user **js** creates a *.forward* file that contains the line:

```
\js, "|/usr/ucb/vacation js"
```

so that one copy of the message is sent to the user, and another is piped into the *vacation*(1) program.

SEE ALSO

newaliases(8), dbm(3X), sendmail(8), uucp(1C), vacation(1)

System Administration for the Sun Workstation

BUGS

Because of restrictions in *dbm*(3X) a single alias cannot contain more than about 1000 characters. Nested aliases can be used to circumvent this limit.

Notes

Notes

Corporate Headquarters
Sun Microsystems, Inc.
2250 Garcia Avenue
Mountain View, CA 94043
415 960-1300
TLX 287815

For U.S. Sales Office locations, call:
800 821-4643
In CA: 800 821-4642

European Headquarters
Sun Microsystems Europe, Inc.
Sun House
31-41 Pembroke Broadway
Camberley
Surrey GU15 3XD
England
0276 62111
TLX 859017

Australia: 61-2-436-4699
Canada: 416 477-6745
France: (1) 46 30 23 24
Germany: (089) 95094-0
Japan: (03) 221-7021
The Netherlands: 02155 24888
UK: 0276 62111

Europe, Middle East, and Africa, call European Headquarters:
0276 62111

Elsewhere in the world, call Corporate Headquarters:
415 960-1300
Intercontinental Sales