Part Number 800-1107-01
Revision: E of 7th January 1984
For: Sun System Release 1.1

User's Manual

for the

Sun Workstation

Sun Microsystems, Inc.,
2550 Garcia Avenue
Mountain View
California 94043
(415) 960-1300

Trademarks

Multibus is a trademark of Intel Corporation.

Sun Workstation is a trademark of Sun Microsystems Incorporated.

UNIX is a trademark of Bell Laboratories.

Revision History

Revision	Date	Comments
A	23 February 1983	First release of this Manual.
B	15 April 1983	Second Release of this manual involved many corrections to manual pages.
C	1 August 1983	Third Release of this manual involved many corrections to manual pages.
D	1 November 1983	Fourth Release of this manual involved many corrections to manual pages. Added OPTIONS to manual pages for clarity. Fixed numerous incorrect cross-references.
E	7 January 1984	Fifth Release of this manual involved many corrections to manual pages.

PERMUTED INDEX

Introduction to the Sun UNIX System Manuals

Table of Contents

INTRODUCTION TO THE SUN UNIX SYSTEM MANUALS

These manuals describe the features of the UNIX† system for the Sun Workstation, and provide tutorial information about the UNIX operating system, its facilities, and its implementation.

There are three major groups of manuals. The first group consists of three manuals which are essentially reference manuals for, respectively, the user, programmer, and system administrator of the Sun UNIX System. These manuals are extensively reworked, revised, reorganized derivations of the original *UNIX Programmer's Manual*; we place them in a unique category because of certain formatting and organizational conventions which have survived their revision. These conventions are explained below. The second group is a new set of reference manuals treating functional areas of the Sun UNIX operating system — windows, system internals, and so on. The final group is an set of manuals which contain tutorial-style papers on various aspects of the system; there is a manual on text editing and documentation, one on C programming, one which introduces UNIX to the first-time user, and so on. In the following, we discuss each group of manuals in more detail.

1. Manuals Derived from the Old UNIX Programmer's Manual

The material in the three Sun reference manuals was originally bound together in a single volume known as the *UNIX Programmer's Manual*. We have unbundled the material into more focused and manageable guides; however, we have maintained the section numbering from the original manual (so that users familiar with it won't be at sea with the new documentation). We hope that new UNIX users will forgive the rather strange numbering scheme that resulted from this compromise, and that older ones will not grumble too much at the demise of the *Programmer's Manual*. The three new manuals are as follows:

1.1. User's Manual

documents the facilities available to a Sun Workstation user (Section 1). The manual also includes a section on games (Section 6), and a section with tables of information about character codes, macro packages for typesetting, and so on (Section 7).

Commands generally reside in the */bin* directory (for *bin*ary programs). Some commands also reside in */ usr/ bin*, or in */ usr/ ucb*, to save space in */bin*. The command interpreters (known as the *shells*) search these directories automatically.

† UNIX is a trademark of Bell Laboratories.

1.2. System Interface Manual

covers areas of interest to programmers. It contains sections on system calls (Section 2), library functions (Section 3), devices and special files (Section 4), and file formats (Section 5). It also includes the *System Interface Overview*, a document which summarizes the facilities — basic kernel functions and standard system abstractions for files and file systems, communications, process control and debugging — provided by the 1.0 version of the Sun UNIX operating system. Finally, a *Tutorials* section contains a paper on the Interprocess Communication facilities included in the Sun UNIX operating system.

System calls are entries into the UNIX system supervisor. The system call interface is identical to a C language procedure call; the equivalent C procedures are described in Section 2. Assorted subroutines are available; they are described in Section 3. The primary libraries in which they are kept are described in *intro*(3). The functions are described in terms of C, but most work with FORTRAN as well. The special files section (4) discusses the characteristics of each system 'file' that actually refers to an I/O device. The names in this section refer to the UNIX system device names for the hardware, instead of the names of the special files themselves. The file formats and conventions section (5) documents the structure of particular kinds of files; for instance, the form of the output of the loader and assembler is given. Files used by only one command are not documented; the assembler's intermediate files, for example, are excluded.

1.3. System Manager's Manual for the Sun Workstation

contains information needed to install and configure the Model 100U, 150U, and 120 Sun Workstations, install the UNIX system, and run the UNIX system on a day-to-day basis. The System Manager's Manual includes a detailed guide to system set-up and operation: *System Installation and Maintenance Guide*, maintenance commands and procedures (Section 8), and tutorials on subjects such as *sendmail* and *uucp* installation.

The maintenance section discusses commands and procedures used by the system manager (who is also called the *super-user*). Most commands and files described are kept in the */etc* directory.

2. Layout of the Reference Manual Pages

Each section of the reference manuals consists of a number of independent entries of a page or so each. The name of the entry is in the upper corners of its pages, followed by the section number, and sometimes a letter characteristic of a subcategory; for instance, graphics is 1G, and the math library is 3M. Entries within each section are in alphabetical order. At the bottom left of each page is the release level of the Sun system, and the date in the middle of the page reflects the date on which the manual page was changed for any reason.

All manual entries have a common format, though for any given entry only relevant subsections appear:

Subsection	*Contains*
NAME	The *name* subsection lists the exact names of the commands and subroutines covered under the entry and gives a very short description of their purpose.
SYNOPSIS	Summarizes the use of the program. A few conventions are used, particularly when commands are being described: **Boldface** words or letters are literals; type them just as they appear. Square brackets '[ ]' around an argument indicate that the argument is optional. When an argument is given as 'name' or 'file', it always refers to a file name. Braces '{ }' around an argument indicate that the argument is mandatory. Ellipses '..' are used to show that the previous argument-prototype may be repeated.
DESCRIPTION	Discusses what the utility does for you.
OPTIONS	Lists the options which apply to the command.
EXAMPLES	Illustrates typical use of the command.
FILES	Lists the names of files which the program uses in the course of its job.
SEE ALSO	Indicates related information — such as other manual entries to read.
DIAGNOSTICS	Deals mainly with the *exit codes* returned to the shell by exiting commands. Exit codes can be used in shell scripts to control the actions of the shell.
BUGS	Documents known problems with the command, sometimes discusses deficiencies, and occasionally describes suggested fixes.

3. New Sun Reference Manuals

There are some original reference manuals written for specific Sun products. These additional manuals are described in the sections below.

3.1. Suncore Programmer's Reference Manual

Reference material for the SunCore Graphics package. In addition to a formal description of all SunCore library calls, there are also descriptions of the Sun device support, and descriptions of the interfaces to the SunCore library via FORTRAN and Pascal.

3.2. Programmer's Reference Manual for SunWindows

Reference and tutorial material for programming applications using the SunWindows facilities.

3.3. System Internals Manual

The *System Internals Manual* discusses several aspects of the implementation of the Sun UNIX System. Contents are:

Networking Implementation Notes
: Describes the internal structure of the networking facilities of the Sun UNIX operating system.

Changes to the Kernel
: Summarizes the changes to the kernel made between the last and the current release. Treats broad, organizational changes and consequences for individual files.

Device Driver Reference Manual for the Sun Workstation UNIX System
: A guide to writing your own device driver. Describes how to rebuild the UNIX system kernel and add software drivers for new devices to the kernel, provides sample drivers, and summarizes functions and interfaces between the driver and the kernel.

Using ADB to Debug the UNIX Kernel
: Describes extensions made to the 0.4 release of the Sun UNIX system debugger, ***adb***, to allow standard ***adb*** commands to function properly with the kernel, and introduces the basics of writing ***adb*** command scripts.

A Fast File System for UNIX
: Describes the reimplementation of the UNIX file system.

The CPU PROM Monitor
: Information on the PROM Monitor functions, commands, messages, and keyboard control codes.

4. Supplementary Manuals

In addition to the manuals derived from the old UNIX Programmer's Reference Manual, there are a number of other manuals ranging in level from basic 'getting started' to technical papers derived from Bell Labs or Berkele publications. These additional manuals are described in the sections below.

4.1. Beginner's Guide to the Sun Workstation

A general overview and introduction to UNIX. The manual provides roadmaps and guides to the rest of the documentation, plus some tutorial introductions to the Sun system. It also contains a general guide to the communication facilities that UNIX offers you. First-time users are advised to start here. The contents are:

Getting Started
: An introduction to the most basic use of the Sun UNIX system. Includes discussions of how to log in, UNIX file and directory structure, basic commands, and how to log out.

Working with Files
: Describes various commands and utilities for file manipulation — ***more***, ***head***, ***tail***, ***grep***, pattern matching utilities, ***sort***, ***diff***, and so on.

Using the Shell
: How to use the facilities of the shell to make your work easier. Covers redirection of input and output, pipes, job control, and the ***history*** and ***alias*** facilities.

Creating and Editing Text Files — The vi Editor
: An introduction to the visual editor.

Printing and Formatting Documents
: How to display and print files, and how to 'turn them into' documents. This chapter covers ***nroff*** and ***troff***, the formatting macro package ***ms***, and the more specialized document formatting preprocessors, ***tbl*** and ***eqn***.

Communications
: Introduces basic UNIX communications facilities: how to use the ***mail*** system, how to ***write*** to other users logged in to the same system, how to use network facilities, ***tip*** and *UUCP*.

Using the C Shell.
: Introducing a popular command interpreter, the C Shell, and many of the commonly used UNIX commands.

Using the Bourne Shell
: The Bourne Shell is the Version 7 UNIX Shell and is heavily used in system command scripts.

Mail User's Guide
: Complete details on the electronic mail facilities. Includes a quick-reference sheet.

Network News User's Guide
: Describes what the network news is, how to join news groups, how to read news, and how to post your own news. Includes a quick-reference sheet.

4.2. Editing and Text Processing

Documents the facilities that UNIX offers for massaging text files in various ways, and for processing documentation. Part One of the manual provides information on the text editors and Part Two describes the document formatting tools. Contents:

An Introduction to Text Editing
Describes the basics of text editing and provides a guide to the available editing tools. Newcomers should start here.

Using the 'vi Visual Display Editor
Tutorial and reference information on the *vi* editor. Includes a quick reference sheet.

Command Reference for the 'ex' Line Editor
A command reference for the *ex* and *vi* editors. Also includes a quick reference sheet.

Using the 'ed' Line Editor
Provides a user's guide to the original UNIX editor.

Using 'sed', the Stream Text Editor
A variant of *ed* for processing large files.

Pattern Scanning and Processing with AWK
Makes it easy to specify many data transformation and selection operations.

An Introduction to Text Processing
Introduces the basics of text processing, macros and macro packages. Includes explanations of the tools and examples of usage.

Formatting Documents with the -ms Macro Package
Describes *-ms*, a standardized package of formatting requests that can be used to lay out most documents. Includes a quick-reference summary of requests.

Formatting Documents with nroff and troff
Introduces the basic document layout facilities. Includes both a user's guide and reference material.

Formatting tables with tbl
Describes *tbl*, an easy-to-learn notation for specifying tabular layout for typesetting.

Typesetting Mathematics with eqn
Describes *eqn*, another easy-to-learn language for doing high-quality mathematical typesetting.

Formatting Documents with the -me Macro Package
A popular macro package for *nroff*. Includes a user's guide, reference material, and a quick-reference summary of requests.

4.3. Programming Tools

Contains information of general interest to anyone using UNIX to write programs:

Programming the Shells
The C-Shell or the Bourne Shell can often be used to create tools without the need to write programs.

UNIX Programming
Describes the programming interface to the UNIX operating system and the standard I/O.

Lint, A C Program Checker
Checks C programs for syntax errors, type violations, portability problems, and a variety of probable errors.

Make — A Program for Maintaining Computer Programs
Indispensable tool for making sure that large programs are properly compiled with minimal effort.

Source Code Control System
A guide to the Source Code Control System (SCCS) — a system for maintaining and controlling multiple versions of text files.

DC — An Interactive Desk Calculator
A superb HP calculator, if you don't need floating point.

BC — Arbitrary Precision Desk-Calculator Language
A front end for DC that provides infix notation, control flow, and built-in functions.
The M4 Macro Processor
M4 is a macro processor useful as a front end for C, Ratfor, Cobol, and in its own right.
LEX — A Lexical Analyzer Generator
Creates a recognizer for a set of regular expressions; each regular expression can be followed by arbitrary C code which will be executed when the regular expression is found.
YACC — Yet Another Compiler Compiler
Converts a BNF specification of a language and semantic actions written in C into a compiler for the language.
Assembler Manual for the Sun Workstation
The ultimate dead language.

4.4. Fortran and Pascal

Information specific to the FORTRAN and Pascal programming languages. Contents:
Fortran Programmer's Guide
Describes the compiler and run-time system for the new extended language, Fortran 77. Also describes the revised I/O library for Fortran 77.
Ratfor — A Preprocessor for a Rational Fortran
Converts a Fortran with C-like control structures and cosmetics into real, ugly Fortran.
System Interface Routines
FORTRAN interfaces to the basic UNIX supervisor facilities.
Pascal User's Manual
Describes the Berkeley Pascal system.

NAME

intro – introduction to commands

DESCRIPTION

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

(1) Commands of general utility.

(1C) Commands for communication with other systems.

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

SEE ALSO

Section 6 for computer games.

Getting Started in the *Beginner's Guide to the Sun Workstation.*

DIAGNOSTICS

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

NAME

adb – debugger

SYNOPSIS

adb [–w] [–k] [–Idir] [objfil [corfil]]

DESCRIPTION

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

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

OPTIONS

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

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

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

USING ADB

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

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

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

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

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

EXPRESSIONS

. The value of *dot*.

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

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

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

integer A number. The prefixes 0o and 0O ("zero oh") force interpretation in octal radix; the prefixes 0t and 0T force interpretation in decimal radix; the prefixes 0x and 0X force interpretation in hexadecimal radix. Thus 0o20 = 0t16 = 0x10 = sixteen. If no prefix appears, then the *default radix* is used; see the $d command. The default radix is initially hexadecimal. The hexadecimal digits are 0123456789abcdefABCDEF with the obvious values. Note that a hexadecimal number whose most significant digit would otherwise be an alphabetic character must have a 0x (or 0X) prefix (or a leading zero if the default radix is hexadecimal).

integer.fraction
A 32 bit floating point number.

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

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

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

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

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

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

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

Unary operators

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

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

- *exp* Integer negation.

~ *exp* Bitwise complement.

*exp* Logical negation.

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

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

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

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

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

e1 **+** *e2* Integer addition.

e1 **-** *e2* Integer subtraction.

e1 ***** *e2* Integer multiplication.

e1 **%** *e2* Integer division.

e1 **&** *e2* Bitwise conjunction.

e1 **|** *e2* Bitwise disjunction.

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

VARIABLES

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

0 The last value printed.

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

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

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

COMMANDS

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

The verbs are:

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

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

= Print the value of *address* itself.

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

: Manage a subprocess.

$ Execute miscellaneous commands.

> Assign a value to a variable or register.

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

! Call the shell to execute the following command.

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

[?, /, Q, =] Modifiers

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

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

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

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

x	2	Print 2 bytes in hexadecimal.
X	4	Print 4 bytes in hexadecimal.
u	2	Print as an unsigned decimal number.
U	4	Print long unsigned decimal.
f	4	Print the 32 bit value as a floating point number.
F	8	Print double floating point.
b	1	Print the addressed byte in octal.
c	1	Print the addressed character.
C	1	Print the addressed character using the standard escape convention: print control characters as ^X and the delete character as ^?.
s	*n*	Print the addressed characters until a zero character is reached.
S	*n*	Print a string using the ^*X* escape convention (see **C** above). *n* is the length of the string including its zero terminator.
Y	4	Print 4 bytes in date format (see *ctime*(3)).
i	n	Print as machine instructions. *n* is the number of bytes occupied by the instruction. In this format, variables 1 and 2 are set to the offset parts of the source and destination respectively.
z	n	Print as machine instructions with 68010 instruction timings. *n* is the number of bytes occupied by the instruction. In this format, variables 1 and 2 are set to the offset parts of the source and destination respectively.
I	0	Print the source text line specified by *dot* (**@** command) or most closely corresponding to *dot* (**?** command).
a	0	Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.
		/ local or global data symbol
		? local or global text symbol
		= local or global absolute symbol
p	4	Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
A	0	Print the value of *dot* in sourcefile symbolic form, that is: *"filename" + nnn.* (Works only if the program has been compiled using the **–go** flag. See *cc*(1).)
P	4	Print the addressed value in sourcefile symbolic form, that is: *"filename" + nnn.* (Works only if the program has been compiled using the **–go** flag. See *cc*(1).)
t	0	When preceded by an integer, tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.
r	0	Print a space.
n	0	Print a newline.
"..."	0	Print the enclosed string.
^		*Dot* is decremented by the current increment. Nothing is printed.
+		*Dot* is incremented by 1. Nothing is printed.
–		*Dot* is decremented by 1. Nothing is printed.

[?, /] Modifiers

[?/]l *value mask*

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

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

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

: Modifiers

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

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

d Delete breakpoint at *address*.

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

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

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

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

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

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

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

k Terminate the current subprocess, if any.

$ Modifiers

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

<<*file* Similar to **<**, but can be used in a file of commands without closing the file.

Variable *9* is saved during the execution of this command, and restored when it completes. There is a (small) finite limit to the number of << files that can be open at once.

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

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

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

b Print all breakpoints and their associated counts and commands.

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

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

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

e Print the names and values of external variables.

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

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

o All integers input are regarded as octal.

q Exit from *adb*.

v Print all non zero variables in octal.

m Print the address map.

f Print a list of known source file names.

p Print a list of known procedure names.

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

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

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

ADDRESS MAPPING

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

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

$$b1 \leq address < e1 \Rightarrow file\ address = address + f1 - b1, \text{ otherwise,}$$

$$b2 \leq address < e2 \Rightarrow file\ address = address + f2 - b2,$$

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

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

FILES

a.out
core

SEE ALSO

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

DIAGNOSTICS

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

BUGS

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

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

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

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

NAME

addbib – create or extend bibliographic database

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

%O UNX 4.3.5.

FILES

promptfile optional file to define prompting

SEE ALSO

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

NAME

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

SYNOPSIS

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

DESCRIPTION

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

OPTIONS

–c *center screen*

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

–m *center screen*

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

–l *side screen*

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

–r *side screen*

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

–t *side screen*

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

–b *side screen*

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

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

EXAMPLE

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

FILES

/usr/suntool/adjacentscreens

SEE ALSO

suntools(1), login(1)

BUGS

Window systems on the screens have to be initialized before running *adjacentscreens*. It is better to run this program just once at boot time in */etc/rc.local* for example, as physical screen positions rarely change

NAME

admin – create and administer SCCS files

SYNOPSIS

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

DESCRIPTION

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

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

OPTIONS

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

-n A new SCCS file is being created.

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

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

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

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

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

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

-a *login*

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

-e *login*

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

-y[*comment*]

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

-m[*mrlist*]

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

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

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

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

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

FLAGS

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

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

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

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

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

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

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

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

<list>

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

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

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

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

m *module*

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

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

v [*program*]

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

FILES

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar – archive and library maintainer

SYNOPSIS

ar drqtpmx [vualbclo] [posname] afile name ...

DESCRIPTION

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

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

OPTIONS

d Delete the named files from the archive file.

r Replace the named files in the archive file.

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

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

p Display the named files in the archive.

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

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

MODIFIERS

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

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

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

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

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

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

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

i Identical to the **b** modifier.

FILES

/tmp/v* temporaries

SEE ALSO

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

BUGS

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

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

NAME

as – mc68000 assembler

SYNOPSIS

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

DESCRIPTION

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

All undefined symbols in the assembly are treated as global.

The output of the assembly is left in the file *objfile*. If that is omitted, and the source file name has the form *xxx.s*, then file *a.out* is used.

OPTIONS

–d2 Specifies that instruction offsets which involve forward or external references, and which have sizes unspecified in the assembly language are two bytes long. The default is four bytes. See also **–J**.

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

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

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

FILES

/tmp/as* default temporary file

SEE ALSO

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

BUGS

Should assemble standard input with no arguments.

The Pascal compiler (*pc*(1)) qualifiles a nested procedure name by chaining the names of the enclosing procedures. This sometimes results in names long enough to abort the assembler, which currently limits identifiers to 50 characters.

NAME

at – execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

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

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

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

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

Standard output or error output is lost unless redirected.

EXAMPLES

Examples of legitimate commands are:

```
at 8a jan 24

at 1530 fr week
```

FILES

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

in /usr/spool/at:

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

SEE ALSO

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

BUGS

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

NAME

awk – pattern scanning and processing language

SYNOPSIS

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

DESCRIPTION

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

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

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

OPTIONS

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

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

LINES, STATEMENTS, AND THE AWK LANGUAGE

Input Lines

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

Pattern-action Statements

A pattern-action statement has the form

```
pattern { action }
```

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

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

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

Format of the Awk Language

Statements are terminated by semicolons, newlines or right braces. An empty expression-list stands for the whole line.

Expressions take on string or numeric values as appropriate, and are built using the operators +, –, *, /, %, and concatenation (indicated by a blank). The C operators ++, —, +=, –=, *=, /=, and %= are also available in expressions.

Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric, providing a form of associative memory. String constants are quoted "...".

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

Built In Functions

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

Patterns

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep*. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

expression *matchop* regular-expression
expression *relop* expression

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

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

Special Variable Names

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

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

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

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

SEE ALSO

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

BUGS

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

NAME

basename – strip filename affixes

SYNOPSIS

basename string [*suffix*]

DESCRIPTION

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

EXAMPLE

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

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

SEE ALSO

sh(1)

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [–c] [–l] [file ...]

DESCRIPTION

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

Comments

are enclosed in /* and */.

Names

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

Other operands

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

Operators

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

Statements

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

Function definitions

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

Functions in –l math library

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

All function arguments are passed by value.

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

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

EXAMPLES

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

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

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

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

OPTIONS

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

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

FILES

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

SEE ALSO

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

BUGS

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

NAME

biff – mail alarm

SYNOPSIS

biff [yn]

DESCRIPTION

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

biff y

enables notification; the command:

biff n

disables it; finally, the command:

biff

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

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

SEE ALSO

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

NAME

/bin/mail – send or receive mail among users

SYNOPSIS

/bin/mail [+] [**–i**] [*person*] ...
/bin/mail [+] [**–i**] **–f** *file*

DESCRIPTION

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

/bin/mail with no argument prints a user's mail, message-by-message, in last-in, first-out order; the optional argument + displays the mail messages in first-in, first-out order. For each message, */bin/mail* reads a line from the standard input to direct disposition of the message.

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

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

COMMANDS

newline Go on to next message.

d Delete message and go on to the next.

p Print message again.

– Go back to previous message.

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

w [*file*] ...
Save the message, without a header, in the named *files* ('mbox' default).

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

EOT (control-D)
Put unexamined mail back in the mailbox and stop.

q Same as EOT.

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

***** Print a command summary.

OPTIONS

+ Display mail messages in first-in, first-out order.

–f *file* Use *file* as if it were the mail file.

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

FILES

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

dead.letter unmailable text

SEE ALSO

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

BUGS

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

Normally anybody can read your mail, unless it is sent by *xsend*(1). An installation can overcome this by making */bin/mail* a set-user-id command that owns the mail directory.

NAME

cal – display calendar

SYNOPSIS

cal [month] year

DESCRIPTION

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

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

Month is a number between 1 and 12.

The calendar produced is that for England and her colonies.

Try September 1752.

NAME

calendar – reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

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

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

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

FILES

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

SEE ALSO

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

BUGS

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

NAME

cat – concatenate and display

SYNOPSIS

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

DESCRIPTION

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

% cat goodies

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

% cat file1 file2 >file3

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

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

OPTIONS

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

–n precedes each line output with its line number.

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

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

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

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

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

SEE ALSO

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

BUGS

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

NAME

cb – C program beautifier

SYNOPSIS

cb

DESCRIPTION

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

NAME

cc – C compiler

SYNOPSIS

cc [**-c**] [**-g**] [**-go**] [**-w**] [**-p**] [**-pg**] [**-O**] [**-R**] [**-fsingle**] [**-fsky**] [**-S**] [**-E**] [**-C**] [**-o** *output*] [**-D** *name=def*] [**-D** *name*] [**-U** *name*] [**-I** *dir*] [**-B** *string*] [**-t**[**p012**]] file ...

DESCRIPTION

Cc is the UNIX C compiler which translates programs written in the C programming language into executable load modules, or relocatable binary programs for subsequent loading with the *ld*(1) linker. In addition to the many flag arguments (options) *cc* accepts several types of files: files whose names end with *.c* are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with *.o* substituted for *.c*. The *.o* file is deleted if a single C program is compiled and loaded all at once.

In the same way, files whose names end with *.s* are taken to be assembly source programs and are assembled, producing a *.o* file.

OPTIONS

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

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

-g Have the compiler produce additional symbol table information for *dbx*(1). Also pass the **-lg** flag to *ld*(1).

-go Have the compiler produce additional symbol table information in an older format which was used by the *sdb* debugger and can still be used by the *adb*(1) debugger, which has not yet been converted to the new format of **-g**. Also pass the **-lg** flag to *ld*(1).

-w Suppress warning messages.

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

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

-O Use the object code optimizer to improve the generated code.

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

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

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

-S Compile the named C programs, and leave the assembler-language output on

corresponding files suffixed '.s'.

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

-C Prevent the macro preprocessor from removing comments.

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

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

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

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

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

-t[**p012**]
Find only the designated compiler passes in the files whose names are constructed by a **-B** option. In the absence of a **-B** option, the *string* is taken to be */usr/new/*. The letter/number combinations that can be specified for the **-t** option have the meanings:
p *cpp* — the C preprocessor.
0 *ccom* — both passes of the C compiler but excluding the optimizer.
1 Ignored in this system — this option would be for the second phase of a two-phase compiler but in the Sun system, *ccom* includes both phases.
2 *c2* — the C optimizer.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C-compatible routines. Unless **-c**, **-S**, or **-E** is specified, these programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name *a.out*. The name *a.out* can be overridden with the loader's **-o***name* option.

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/lib/cpp	preprocessor
/lib/ccom	compiler
/usr/c/occom	backup compiler
/usr/c/ocpp	backup preprocessor
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/usr/lib/gcrt0.o	startoff for gprof-profiling
/lib/fcrt0.o	SKY runtime startoff
/lib/fmcrt0.o	SKY startoff for profiling
/usr/lib/fgcrt0.o	SKY startoff for gprof-profiling
/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libc_p.a	profiling library, see *intro*(3)
/usr/include	standard directory for '#include' files
mon.out	file produced for analysis by *prof*(1)

gmon.out file produced for analysis by *gprof*(1)

SEE ALSO

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

DIAGNOSTICS

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

NAME

cd – change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

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

SEE ALSO

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

NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

cdc –rSID [**–m** [mrlist]] [**–y** [comment]] file ...

DESCRIPTION

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

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

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

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

OPTIONS

All the described options apply independently to each named file:

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

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

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

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

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

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

–y[*comment*]

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

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

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

EXAMPLES

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

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

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

does the same thing.

WARNINGS

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

OPTIONS

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

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

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

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

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

FILES

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

SEE ALSO

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

NAME

checknr – check nroff/troff files

SYNOPSIS

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

DESCRIPTION

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

(1) Font changes using \fx ... \fP.

(2) Size changes using \sx ... \s0.

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

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

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

OPTIONS

–s Ignore \s size changes.

–f Ignore \f font changes.

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

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

SEE ALSO

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

BUGS

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

NAME

chgrp – change group

SYNOPSIS

chgrp [**–f**] group file ...

DESCRIPTION

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

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

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

FILES

/etc/group

SEE ALSO

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

NAME

chmod – change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

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

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

A symbolic *mode* has the form:

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

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

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

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

EXAMPLES

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

```
chmod o-w file
chmod +x file
```

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

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

SEE ALSO

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

NAME

chsh – change default login shell

SYNOPSIS

chsh username [shell]

DESCRIPTION

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

EXAMPLES

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

SEE ALSO

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

NAME

clear – clear workstation or terminal screen

SYNOPSIS

clear

DESCRIPTION

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

FILES

/etc/termcap **terminal capability data base**

NAME

cmp – compare two files

SYNOPSIS

cmp [**–l**] [**–s**] file1 file2

DESCRIPTION

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

OPTIONS

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

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

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

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

NAME

col – filter reverse paper motions

SYNOPSIS

col [–bfx]

DESCRIPTION

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

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

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

OPTIONS

–f Fine: although *col* accepts half line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full line boundary. The **–f** option suppresses this treatment — in this case the output from *col* may contain forward half line feeds (ESC-9), but will still never contain either kind of reverse line motion.

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

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

SEE ALSO

troff(1), tbl(1)

BUGS

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

NAME

colcrt – filter nroff output for CRT previewing

SYNOPSIS

colcrt [–] [–2] [file ...]

DESCRIPTION

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

OPTIONS

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

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

EXAMPLE

A typical use of *colcrt* would be

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

SEE ALSO

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

BUGS

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

Can't back up more than 102 lines.

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

Lines are trimmed to 132 characters.

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

NAME

colrm – remove columns from a file

SYNOPSIS

colrm [startcol [endcol]]

DESCRIPTION

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

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

Column numbering starts with column 1.

SEE ALSO

expand(1)

NAME

comb – combine SCCS deltas

SYNOPSIS

/usr/sccs/comb [**–o**] [**–s**] [**–p** *sid*] [**–c** *list*] file ...

DESCRIPTION

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

OPTIONS

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

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

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

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

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

100 * (original – combined) / original

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

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

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [– [123]] file1 file2

DESCRIPTION

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

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

SEE ALSO

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

NAME

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

SYNOPSIS

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

DESCRIPTION

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

... | compact | uncompact | ...

operates as a (very slow) no-op.

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

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

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

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

FILES

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

SEE ALSO

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

NAME

cp – copy files

SYNOPSIS

cp [–i] [–r] file1 file2

cp [–i] [–r] file ... directory

DESCRIPTION

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

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

Cp refuses to copy a file onto itself.

OPTIONS

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

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

EXAMPLES

To make a backup copy of *goodies*:

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

To copy an entire directory hierarchy:

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

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

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

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

SEE ALSO

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

BUGS

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

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio **–o** [**acBvs**]

cpio **–i** [**Bcdmrtuv6s**] [patterns]

cpio **–p** [**adlmruvs**] directory

DESCRIPTION

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

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

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

OPTIONS

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

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

d *Directories* should be created as needed.

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

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

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

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

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

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

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

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

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

EXAMPLES

To copy the contents of a directory into an archive:

```
% ls | cpio –o > /dev/mt0
```

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

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

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

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

SEE ALSO

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

BUGS

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

NAME

cpp – the C language preprocessor

SYNOPSIS

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

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. Use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc*(1) command since the functionality of *cpp* may someday be moved elsewhere. See *m4*(1) for a general macro processor.

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

OPTIONS

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

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

–U*name*

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

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

–D*name*

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

–D*name=def*

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

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

–R Allow recursive macros.

CPP DIRECTIVES

All *cpp* directives start with lines begun by #. The directives are:

#define *name token-string*

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

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

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

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

#include *"filename"*
#include *<filename>*
Include at this point the contents of *filename* (which is then run through *cpp*). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See the –I option above for more detail.

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

#endif
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

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

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

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

#else Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output. And vice versa.

The test directives and the possible **#else** directives can be nested.

DETAILS OF THE C PREPROCESSOR

Directory search order for #include files is:

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

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

An unescaped newline (the single character "0) terminates a character constant or quoted string.

An escaped newline (the two-character sequence "\0) may be used in the body of a '#define' statement to continue the definition onto the next line. The escaped newline is not included in the macro body.

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

```
left/* la di da */right
```

may expand 'left' and 'right' but will never expand 'leftright'. If neither 'left' nor 'right' is a macro then the output is 'leftright', even if 'leftright' is defined as something else. The file:

```
#define agelimit(a,b)b/**/a
agelimit(1,2)
```

produces '21' because the comment causes a break enabling *cpp* to recognize 'b' and 'a' as formals in the string 'b/**/a'.

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

```
#define foo(a) "
foo(bar)
```

is the seven characters " '\bar'". Macro names are not recognized inside character constants or quoted strings during the regular scan. Thus:

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

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

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

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

produces 'foo'. The token appearing immediately after a '#ifdef' or '#ifndef' is not expanded (of course!).

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

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

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

Incompatibility: The virgule '/' in 'a=/*b' is interpreted as the first character of the pair '/*' which introduces a comment, rather than as the second character of the divide-and-replace operator '=/'. This incompatibility reflects the recent change in the C language which made 'a/=*b' the legal way to write such a statement if the meaning 'a=a/ *b' is intended.

FILES

/usr/include standard directory for #include files

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

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

NOTES

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

NAME

crypt – encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

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

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

will print the contents of *clear.file*.

Files encrypted by *crypt* are compatible with those treated by the editor *ed*(1) in encryption mode.

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

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

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

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

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(8)

RESTRICTIONS

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

NAME

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

SYNOPSIS

csh [–cefinstvVxX] [arg ...]

DESCRIPTION

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

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

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

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

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

Lexical structure

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

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

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

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed.

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

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

Jobs

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

[1] 1234

indicating that this job is job number 1 and has one (top-level) process, whose process id is 1234.

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

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

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

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

Status reporting

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

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

Substitutions

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

History substitutions

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

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

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

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

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

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

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

#	the entire command line typed so far
0	first (command) word
n	*n*'th argument
^	first argument, that is, '1'
$	last argument
%	word matched by (immediately preceding) ?*s*? search
x-*y*	range of words
-*y*	abbreviates '0-*y*'
*	abbreviates '^-$', or nothing if only 1 word in event
*x**	abbreviates '*x*-$'
x-	like '*x**' but omitting word '$'

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

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

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

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

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

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

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

Quotations with ' and "

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

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

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, the argument list is left unchanged.

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

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

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

Variable substitution

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

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

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

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

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

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

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

$name
${name}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Command and filename substitution

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

Command substitution

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

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

Filename substitution

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

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

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

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

Input/output

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

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

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

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

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

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

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

given. Otherwise similar to '>'.

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

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

Expressions

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

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

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

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

@ grab = $grab + 1

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

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

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

Control flow

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

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

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

Builtin commands

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

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

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

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

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

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

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

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

is tried to see if it is a directory.

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

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

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

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

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

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

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

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

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

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

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

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

goto word

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

hashstat

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

history
history *n*
history **-r** *n*
history **-h** *n*

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

if (expr) command

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

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

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

jobs
jobs **-l**

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

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

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

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

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

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

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

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

login

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

logout

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

nice
nice +number
nice command
nice +number command

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

nohup
nohup command

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

notify
notify %job ...

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

onintr
onintr -
onintr label

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

In any case, if the shell is running detached and interrupts are being ignored, all forms of

onintr have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

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

pushd
pushd name
pushd + n
With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (as in *cd)* and pushes the old current working directory (as in *csw)* onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

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

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

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

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

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

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

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

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

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

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw
Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '[...]' may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

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

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

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

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

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

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

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

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

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

%job
Brings the specified job into the foreground.

%job &
Continues the specified job in the background.

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

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

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

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

Pre-defined and environment variables

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

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is similarly handled; it is only necessary to worry about its setting in the file *.cshrc*, as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it. It could be set once in the *.login* except that commands through the network would not see the definition.

argv Set to the arguments to the shell. Positional parameters are substituted from

this variable: '$1' is replaced by '$argv[1]', etc.

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

cwd The full pathname of the current directory.

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

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

history Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. If you use an overly-large value for *history*, the shell may run out of memory. The last executed command is always saved on the history list.

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

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

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

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

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

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

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

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

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

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable, only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the superuser the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the –c nor the –t option will normally hash the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be

found.

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

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

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

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

time Controls automatic timing of commands. The **time** variable can be supplied with one or two values, such as 'set time=3' or 'set time=(3 "%%E %/usr/src/man/man1/SCCS/s.csh.1%")'. The first value is a number — *n* for instance. The shell displays a resource-usage summary for any command running for more than *n* CPU seconds. The second value is optional and is a character string which determines which resources the user wishes displayed. The character string can be any string of text with embedded control key-letters in it. A control key-letter is a percent sign (%) followed by a single *upper-case* letter. To print a percent sign, use two percent signs in a row. Unrecognized key-letters are simply printed. The control key-letters are:

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

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

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

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

verbose Set by the –v command line option, causes the words of each command to be

printed after history substitution.

Non-builtin command execution

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

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

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

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

Argument list processing

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

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

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

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

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

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

–s Command input is taken from the standard input.

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

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

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

–V Causes the *verbose* variable to be set even before '.cshrc' is executed.

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

After processing of flag arguments if arguments remain but none of the **–c**, **–i**, **–s**, or **–t** options was given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by '$0'. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with

this shell, the shell will execute such a 'standard' shell if the first character of a script is not a '#', that is, if the script does not start with a comment. Remaining arguments initialize the variable *argv*.

Signal handling

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

FILES

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

LIMITATIONS

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

SEE ALSO

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

BUGS

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

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

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

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

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

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

Quoting conventions are contradictory and confusing.

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

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

There is no way to direct error output to one place and standard output to another place.

NAME

ctags – create a tags file

SYNOPSIS

ctags [–BFatuwvx] file ...

DESCRIPTION

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

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

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

OPTIONS

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

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

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

–a append to tags file.

–t create tags for typedefs.

–w suppress warning diagnostics.

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

FILES

tags output tags file

SEE ALSO

ex(1), vi(1)

BUGS

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

Does not know about #ifdefs.

NAME

date – display or set the date

SYNOPSIS

date [–u] [yymmddhhmm [.ss]]

DESCRIPTION

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

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

OPTIONS

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

EXAMPLE

```
date 10080045
```

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

FILES

/usr/adm/wtmp to record time-setting

SEE ALSO

utmp(5)

DIAGNOSTICS

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

NAME

dbx – debugger

SYNOPSIS

dbx [**–r**] [**–l**] [**–I** *dir*] [*objfile*] [*coredump*]

DESCRIPTION

Dbx is a tool for source level debugging and execution of programs under the UNIX operating system. *Objfile* is an object file produced by a compiler with the appropriate flag (**–g**) specified to produce symbol information in the object file. IMPORTANT: every stage of the compile process, including the *ld* phase, should include the **–g** option. Currently, *cc*(1) and *f77*(1) produce the appropriate source information.

If no *objfile* is specified, *dbx* looks for a file named *a.out* in the current directory. The object file contains a symbol table which includes the names of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

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

Debugger commands in the file *.dbxinit* are executed if that file exists either in the current directory, or in the user's home directory if *.dbxinit* doesn't exist in the current directory.

OPTIONS

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

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

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

Dbx just prompts and waits for a command unless **–r** is specified.

DBX INTERFACE

The *dbx* interface for debugging programs consists of several groups of commands:

Execution and Tracing
: Runs the program being debugged, traces its execution, and stops at selected places.

Displaying and Naming Data
: Displays data and locates variables in the debugged program.

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

Machine Level Commands
: Provides access to memory locations and machine registers.

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

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

Expressions

Dbx Expressions are combinations of variables, constants, procedure calls, and operators. Precede hexadecimal constants by a '0x' and octal constants by a '0'. Enclose character constants in single quotes. Expressions cannot involve strings, structures, or arrays, although elements of structures or arrays may be used.

Dbx recognizes these operators:

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

<< >>
left shift and right shift.

& *
address of operator, and contents of operator.

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

&& ||
logical and, and logical or.

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

Precedence and associativity of operators are the same as C. Parentheses can be used to change precedence of operators.

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

Dbx and Fortran

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

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

Dbx Scope Rules

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

'Func' is used for name resolution, as in the command: **print grab** where grab may be defined in two different routines. The search order (for C and Fortran programs at the time of writing) is:

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

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

'File' is used to:

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

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

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

1. Execution and Tracing Commands

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

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

Source-line-number is evaluated relative to the current 'file' and must be within the current procedure/function or the stack will be incorrect and unexpected behavior will result. Execution cannot be continued if the process has 'finished', that is, called the standard procedure 'exit'. *Dbx* does not allow the process to exit, thereby letting the user examine the program state.

trace [**in** *procedure/function*] [**if** *condition*]
trace *source-line-number* [**if** *condition*]
trace *procedure/function* [**if** *condition*]
trace *expression* **at** *source-line-number* [**if** *condition*]
trace *variable* [**in** *procedure/function*] [**if** *condition*]
Display tracing information when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

If no argument is specified, all source lines are displayed before they are executed. Execution is substantially slower during this form of tracing.

The clause '**in** *procedure/function*' restricts tracing information to be displayed only while executing inside the given procedure or function. Note that the '*procedure/function*' traced

must be visible in the scope in which the trace command is issued — see the '*func*' command.

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

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

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

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

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

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

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

stop if *condition*
stop at *source-line-number* [**if** *condition*]
stop in *procedure/function* [**if** *condition*]
stop *variable* [**if** *condition*]
: Stop execution when the given line is reached, procedure or function called, variable changed, or condition true.

when in *procedure/function* { *command*; ... }
when at *source-line-number* { *command*; ... }
when *Boolean condition* { *command*; ... }
: Execute the given *dbx commands* when the procedure or function is called, line number is reached, or Boolean condition is true. The braces and semicolon, separating each command, are mandatory.

status [> *filename*]
: Display the currently active **trace** and **stop** commands.

delete *command-number*
: Remove the trace or stop corresponding to the given number. The **status** command displays the numbers associated with traces and stops.

catch *number*
ignore *number*
: Start or stop trapping signal *number* before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. Initially all signals are trapped except SIGHUP, SIGCONT, SIGCHILD, SIGALRM, SIGKILL, SIGSTP, and SIGWINCH.

step Execute one source line.

next Execute up to the next source line. The difference between this and **step** is that if the line

contains a call to a procedure or function **step** stops at the beginning of that routine, while **next** does not.

2. Displaying and Naming Data

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

whatis *name*
Print the declaration of the given name, which may be qualified with block names as above.

which *identifier*
Print the full qualification of the given identifer, that is, the outer blocks that the identifier is associated with.

whereis *identifier*
Print the full qualification of all the symbols whose name matches the given identifier. The order in which the symbols are displayed is not meaningful.

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

call *procedure(parameters)*
Execute the object code associated with the named procedure or function. If the source file that the routine is defined in was compiled with the -g flag, the number and type of parameters must match. However, if C routines are called which are not compiled with the -g flag, *dbx* will not do parameter checking. The parameters are simply pushed on the stack as given in the parameter list. Currently, string and structure parameters are not passed properly for C and parameters greater than four bytes (for example, complex and double precision variables) and alternate return points are not passed properly for Fortran.

where Display a list of the active procedures and functions.

dump [> *filename*]
Show the names and values of all active variables.

3. Accessing Source Files

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

file [*filename*]
Change the current source file name to *filename* (apple.c for instance). Display the current source file name if no *filename* is specified.

func [*procedure/function/program name*]
Change the current function. Display the current function if none is specified. Changing the current function implicitly changes the current source file variable **file** to the one that

contains the function; it also changes the current scope used for name resolution. If the entire scope of the program is desired, the argument should be the object file name.

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

use *directory-list*
Set the list of directories to be searched when looking for source files.

4. Machine Level Commands

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*variable*] [**if** *cond*]
stopi [**at**] [*address*] [**if** *cond*]
Turn on tracing or set a stop using a machine instruction address.

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

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

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

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

5. Miscellaneous Commands

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

alias *new-command-name old-command-name*
Respond to *new-command-name* as though it were *old-command-name.*

help [**cont**] [**trace**] [**stop**] [**when**]
Display a synopsis of *dbx* commands. If an argument is given, the synopsis is restricted to show variations of only that particular command.

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

quit Exit *dbx.*

FILES

a.out	default object file
core	default core file
~/.dbxinit	initial commands

SEE ALSO

cc(1) C compiler
f77(1) Fortran compiler
pc(1) Pascal compiler

COMMENTS

Dbx suffers from a 'multiple include' malady. If you have a program consisting of a number of object files and each is built from source files that include header files, the symbolic information for the header files is replicated in each object file. Since about one debugger start-up is done for each link, having the linker (ld) re-organize the symbol information won't save much time, though it would reduce some of the disk space used. The problem is an artifact of the unrestricted semantics of #include's in C; for example an include file can contain static declarations that are separate entities for each file in which they are included.

Currently the only way to type cast is on the machine level using the *mode* specifier (e.g. **print (char *) p** is not allowed).

Dbx does not correctly handle expressions involving pointers to objects where the pointer is offset by some integer value (for example, *(p-5)).

BUGS

Dbx doesn't handle Fortran entry points well — it treats them as if they were independent routines.

Dbx doesn't handle *assigning* to C bit fields and Fortran complex types correctly (see the *assign/set* command).

The *use* command and -I option don't override the current directory if there is source by the same name in the current directory.

If you link with the -g flag but don't compile anything with it, dbx core dumps when reading in the object file.

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

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

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

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

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

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

d
: The top value on the stack is duplicated.

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

f
: All values on the stack and in registers are printed.

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

x
: treats the top element of the stack as a character string and executes it as a string of dc commands.

X
: replaces the number on the top of the stack with its scale factor.

[... **]**
: puts the bracketed ascii string onto the top of the stack.

<*x* >*x* =*x*
: The top two elements of the stack are popped and compared. Register *x* is executed if they obey the stated relation.

v
: replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

!
: interprets the rest of the line as a UNIX command.

c
: All values on the stack are popped.

i
: The top value on the stack is popped and used as the number radix for further input. **I** pushes the input base on the top of the stack.

o
: The top value on the stack is popped and used as the number radix for further output.

O
: pushes the output base on the top of the stack.

k
: the top of the stack is popped, and that value is used as a non-negative scale factor: the

appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

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

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

; : are used by *bc* for array operations.

EXAMPLE

Print the first ten values of n!

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

SEE ALSO

bc(1), which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

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

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

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

OPTION	*VALUES*
if=*name*	input file is taken from *name*; standard input is default.
of=*name*	output file is taken from *name*; standard output is default. Note that *dd* creates and explicit output file; therefore the **seek** option is usually useless with explicit output except in special cases such as using magnetic tape or raw disk files.
ibs=*n*	input block size *n* bytes (default 512).
obs=*n*	output block size *n* bytes (default 512).
bs=*n*	set both input and output block size, superseding **ibs** and **obs**; also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
files=*n*	copy *n* input files before terminating (makes sense only when input is a magtape or similar device).
seek=*n*	seek *n* records from beginning of output file before copying. This option generally only works with magnetic tapes and raw disk files and is otherwise usually useless if the explicit output file was named with the **of** option.
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
block	convert variable length records to fixed length
unblock	convert fixed length records to variable length
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k** (kilobytes) to specify multiplication by 1024, **b** (blocks of 512 bytes) to specify multiplication by 512, or **w** (words) to specify multiplication by 4; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if **ascii**, **unblock**, **ebcdic**, **ibm**, or **block** conversion is specified. In the first two cases, **cbs** characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and new-line added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size **cbs**.

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

EXAMPLE

To read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x*:

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

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

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

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

BUGS

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

NAME

delta – make a delta (change) to an SCCS file

SYNOPSIS

/usr/sccs/delta [**-r** *SID*] [**-s**] [**-n**] [**-g** *list*] [**-m** [*mrlist*]] [**-y** [*comment*]] [**-p**] file ...

DESCRIPTION

Delta permanently introduces into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

Delta may issue prompts on the standard output depending upon certain options specified and flags (see *admin*(1)) that may be present in the SCCS file (see **-m** and **-y** options below).

OPTIONS

Options apply independently to each named file.

-r *SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this option is necessary only if two or more outstanding *get*'s for editing (**get -e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **-r** option can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

-s Do not display the created delta's SID, number of lines inserted, deleted and unchanged in the SCCS file.

-n Retain the edited *g-file* which is normally removed at completion of delta processing.

-g *list* Specifies a *list* of deltas to be *ignored* when the file is accessed at the change level (SID) created by this delta. See *get*(1) for the definition of *list*.

-m [*mrlist*]

If the SCCS file has the **v** flag set (see *admin*(1)), a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

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

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

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that the MR numbers were not all valid).

-y [*comment*]

Arbitrary text to describe the reason for making the delta. A null string is considered a valid *comment*.

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

-p Display (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*-file are explained in the *Source Code Control System User's Guide.* The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/bin/diff	Program to compute differences between the "gotten" file and the *g-file*.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get*/*delta* sequences should be used.

If the standard input (-) is specified on the *delta* command line, the -m (if necessary) and -y options *must* also be present. Omission of these options is an error.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

deroff – remove nroff, troff, tbl and eqn constructs

SYNOPSIS

deroff [**–w**] file ...

DESCRIPTION

Deroff reads each file in sequence and removes all *nroff* and *troff* command lines, backslash constructions, macro definitions, *eqn* constructs (between '.EQ' and '.EN' lines or between delimiters), and table descriptions and writes the remainder on the standard output. *Deroff* follows chains of included files ('.so' and '.nx' commands); if a file has already been included, a '.so' is ignored and a '.nx' terminates execution. If no input file is given, *deroff* reads from the standard input file.

OPTIONS

–w Generate a word list, one word per line. A 'word' is a string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed. All other characters are ignored.

SEE ALSO

troff(1), eqn(1), tbl(1)

BUGS

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

NAME

df – report free disk space on file systems

SYNOPSIS

df [**–l**] [filesystem ...] [file ...]

DESCRIPTION

Df displays the amount of disk space occupied by each file system, the amount of this space which is used and available, and how much of the file system's total capacity has been used. Used without arguments, *df* produces something like:

```
% df
Filesystem  kbytes  used   avail  capacity  Mounted on
/dev/ip0a    7445   4714   1986   70%       /
/dev/ip0g   42277  35291   2758   93%       /usr
%
```

Note that used+ avail is less than the amount of space in the file system (kbytes); this is because the system reserves a fraction of the space in the file system to allow its file system allocation routines to work well. The amount reserved is typically about 10%; this may be adjusted using *tunefs*(8). When all the space on a file system except for this reserve is in use, only the super-user can allocate new files and data blocks to existing files. When a file system is overallocated in this way, *df* may report that the file system is more than 100% utilized.

If arguments to *df* are disk partitions (for example, */dev/ip0a*) or UNIX path names, *df* produces a report on the file system containing the named file. Thus "df ." shows the amount of space on the file system containing the current directory.

OPTIONS

–l Report also the number of used and free inodes.

FILES

/etc/fstab list of normally mounted filesystems

SEE ALSO

du(1), fstab(5), icheck(8), quot(8), tunefs(8)

NAME

diff – differential file and directory comparator

SYNOPSIS

diff [**–cefh**] [**–b**] file1 file2
diff [**–D***string*] [**–b**] file1 file2
diff [**–l**] [**–r**] [**–s**] [ ] [**–S***name*] [**–cefh**] [**–b**] dir1 dir2

DESCRIPTION

diff is a differential file comparator. When run on regular files, and when comparing text files which differ during directory comparison (see the notes below on comparing directories), *diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, *diff* finds a smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, either may be given as '–', in which case the standard input is used. If *file1* is a directory, a file in that directory whose file-name is the same as the file-name of *file2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

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

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

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

If both arguments are directories, *diff* sorts the contents of the directories by name, and then runs the regular file *diff* program as described above on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed.

OPTIONS

Except for **–b**, which may be given with any of the others, the following options are mutually exclusive:

–e produce a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

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

Extra commands are added to the output when comparing directories with **–e**, so that the result is a *sh*(1) script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2*.

–f produces a script similar to that of **–e**, not useful with *ed*, and in the opposite order.

–c produces a diff with lines of context. The default is to present 3 lines of context and may be changed, (to 10, for example), by **–c10**. With **–c** the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from *file1* are marked with '–'; those added to *file2* are marked '+ '. Lines which are changed from one file to the other are marked in both files with '!'.

–h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

Options for the second form of *diff* are as follows:

–D*string*
create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1*, while defining *string* will yield *file2*.

–b ignore trailing blanks (spaces and tabs); other strings of blanks compare equal.

Options when comparing directories are:

–l long output format; each text file *diff* is piped through *pr*(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.

–r apply *diff* recursively to common subdirectories encountered.

–s report files which are the same, which are otherwise not mentioned.

–S*name*
starts a directory *diff* in the middle, beginning with file *name*.

FILES

/tmp/d?????
/usr/lib/diffh for –h
/usr/bin/pr

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff3(1)

DIAGNOSTICS

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

BUGS

Editing scripts produced under the **–e** or **–f** option are naive about creating lines consisting of a single '.'.

When comparing directories with the **–b** option specified, *diff* first compares the files (as in *cmp*), and then decides to run the *diff* algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical, because the only differences are insignificant blank string differences.

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [–ex3] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

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

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

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

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

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

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

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(1)

BUGS

Text lines that consist of a single '.' will defeat –e.

NAME

du – summarize disk usage

SYNOPSIS

du [**–s**] [**–a**] [name ...]

DESCRIPTION

Du gives the number of kilobytes contained in all files and, recursively, directories within each specified directory or file *name*. If *name* is missing, '.' (the current directory) is used.

A file which has multiple links to it is only counted once.

OPTIONS

–s Only display the grand total.

–a Generate an entry for each file.

Entries are generated only for each directory in the absence of options.

EXAMPLE

Here is an example of using *du* in a directory. We used the *pwd* command to identify the directory, then used *du* to show the usage of all the subdirectories in that directory. The grand total for the directory is the last entry in the display:

```
% pwd
/usr/henry/misc
% du
5       ./jokes
33      ./squash
44      ./tech.papers/lpr.document
217     ./tech.papers/new.manager
401     ./tech.papers
144     ./memos
80      ./letters
388     ./window
93      ./messages
15      ./useful.news
1211    .
%
```

SEE ALSO

df(1), quot(8)

BUGS

Non-directories given as arguments (not under **–a** option) are not listed.
If there are too many distinct linked files, *du* counts the excess files multiply.

NAME

echo – echo arguments

SYNOPSIS

echo [–n] [argument ...]

DESCRIPTION

Echo **writes its arguments on the standard output. Arguments must be separated by spaces or tabs, and terminated by a newline.**

Echo **is useful for producing diagnostics in shell programs and for writing constant data on pipes. If you are using the Bourne Shell (*sh*(1)), you can send diagnostics to the standard error file by typing: 'echo ... 1>&2'.**

OPTIONS

–n Don't add the newline to the output.

NAME

ed – text editor

SYNOPSIS

ed [–] [–x] [filename **]**

DESCRIPTION

Ed is the basic text editor in the UNIX system. While *ed* is for all practical purposes superseded by *vi*, *ed* is still used by other system utilities such as SCCS. *Ed* is a line editor — in general, you must specify the line or lines on which to perform the operation you choose (see *Line Addressing*, below, for a discussion of how to form line-addresses for *ed*). You can tell *ed* to perform various operations on the lines. For instance, you can print (display) lines; you can change lines; you can insert new lines into the buffer; you can delete existing lines; you can move or copy lines to a different place in the buffer; you can substitute character strings within lines. See *List of Operations*, below, for a guide. Also, see *Regular Expressions* for string-matching metacharacters.

Ed does not directly change the contents of a file — when editing a file, *ed* reads the contents of the file into a *buffer* or *scratchpad*. All changes made during an editing session are made on the contents of this buffer. The copy must be 'saved' or 'written' — using the *w* (write) operation — to save changes.

The command-line shown in the synopsis above invokes *ed*. If *filename* is given, *ed* reads a copy of *filename* into its buffer so that it can be edited (simulates an *e* operation on *filename*).

Ed commands have a simple and regular structure: commands consists of an optional line-address, or two optional line-addresses separated by a comma, then a single letter operation, optionally followed by some other parameter:

[*line-address* [,*line-address*]] **operation** [*parameter*]

For example, '1,10p' means 'print (display) lines 1 through 10' (two line-addresses), '5a' means 'append after line 5' (one line-address), and d means 'delete the current line' (no line-address with the current line used as default). *Parameter* varies for each operation — for the move and transfer operations, for example, it is the line that the addressed lines are to be moved or transferred after. These operations actually have three line-addresses. For reading and writing a file, *parameter* specifies the name of the file that is to be read.

Ed is extremely terse in its interaction with the user — *ed*'s normal response to just about any problem is simply a question mark **?**. You get this response when, for instance, *ed* can't find a specified line in the buffer, or if a search for a regular expression fails in a substitute (**s**) operation.

OPTIONS

– Suppress the display of character counts normally given by the **e**, **r**, and **w** operations — can be used when the standard input is an editor script.

–x Simulate an **x** operation on the named file before reading it into the buffer, to handle encryption.

LINE ADDRESSING

The format of *ed* operations above shows that an operation can be preceded by one or two line-addresses, both of which are optional. If only one line-address is specified, operations are performed on that specific line. If two line-addresses are supplied, *ed* operates on the inclusive range of lines between them.

Line-addresses are usually separated from each other by a comma — for instance:

1,10p

prints (displays) lines 1 thru 10.

Line addresses may also be separated by a semicolon. Whereas the starting position for line-addresses separated by a comma is the same place in the buffer, when a line-address is followed by a semicolon, the current line is set to the line-address preceding the semicolon before any subsequent line-addresses are interpreted. For example:

/Domaine Chandon/;//p

sets the current line to the first occurrence of the string 'Domaine Chandon' before starting the search for the second occurrence. This feature can be used to determine the starting line for forward and backward searches ('/', '?').

Lines can be accessed (addressed, in *ed* terminology) in several ways, but the most easily understood way of addressing lines is by line number. Line numbers in *ed* are relative to the start of the buffer. In practice, addressing lines by number proves to be the most awkward to use, so *ed* provides other mechanisms for line-addressing. Note that the line numbers associated with lines in the buffer are not physically present with the text of the lines — they are just an addressing mechanism.

While *ed* is working on the buffer, it keeps track of the line on which you last performed some operation. This line is called the 'current line'. As described below, you can indicate the current line by typing a period character (.).

If you don't specify a line for an operation to operate on, most *ed* operations work on the line addressed by the current line.

When *ed* starts working on a file, the current line is positioned at the last line in the buffer. Thereafter, the current line usually changes when any operation is performed. In general, the current line sits at the last line affected by whatever *ed* operation you used. For instance, if you print lines 1 through 10 of the buffer, after the lines are displayed, the current line will be positioned at line 10.

Line-addresses are constructed from elements as shown in the list below. Some special characters are used as a shorthand for certain line-addresses:

. ('dot') addresses the current line.

$ addresses the last line of the buffer.

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

'*x* addresses the line marked with the name *x*, which must be a lower-case letter. Mark lines with the **k** operation described below.

/regular expression/

A *regular expression* enclosed in slashes '/' searches forward from the current line and stops at the first line containing a string that matches the *regular expression*. If necessary, the search wraps around to the beginning of the buffer.

?regular expression?

A *regular expression* enclosed in question marks '?' searches backward from the current line and stops at the first line containing a string that matches the *regular expression*. If necessary the search wraps around to the end of the buffer.

address± *nnn*

An *address* followed by a plus sign '+' or a minus sign '–' followed by a decimal number specifies that line-address plus or minus the indicated number of lines. Plus is assumed if no signs are given.

± *address*

An *address* beginning with '+' or '–' is taken relative to the current line; in other words, '–5' is understood to mean '.–5'.

address±

An *address* ending with '+' or '–', adds or subtracts 1. As a consequence of this rule and the previous rule, the line-address '–' refers to the line before the current line. Moreover, trailing '+' and '–' characters have cumulative effect, so '––' refers to the current line less 2.

^ To maintain compatibility with earlier versions of *ed*, the character '^' in line-addresses is

equivalent to '–'.

Ed operations do not necessarily use line-addresses; they may use one or two. Operations which don't use line-addresses regard the presence of a line-address as an error. Operations which accept one or two line-addresses assume default line-addresses if these are not specified. If more line-addresses are given than such an operation requires, the last one or two (depending on what is accepted) are used. The second line-address of any two-address sequence must be greater than the first line-address — that is, the second line must follow the first line in the buffer.

LIST OF OPERATIONS

Ed operates in one of two major modes: *command mode* and *text input mode*. *Ed* always starts up in command mode.

While you are typing commands at *ed*, you are in command mode. Some commands — **a** for append, **c** for change, and **i** for insert — provide for adding new text to the buffer. While *ed* is accepting new text, you are in text input mode. You exit from text input mode by typing a period '.' alone at the beginning of a line. *Ed* then reverts to command mode. For example, here is a very short illustration of command mode versus text mode:

```
example% ed winelist          (tell ed to edit a file called winelist)
42                            (ed states there are 42 characters in the file)
1,$p                          (in command mode — tell ed to print all lines)
1978  Chateau Chunder
1979  Redeye Canyon
a                             (in command mode — tell ed to append text)
1980  Doomsday Special        (text input mode — add a new line)
.                             (period ends text input mode)
p                             (back in command mode — print last line entered)
1980  Doomsday Special
w                             (command mode — write the file)
65                            (ed displays the number of characters written)
q                             (command mode — quit the edit session)
example%                      (back in the Shell)
```

If you interrupt *ed*, it displays '?interrupted' and returns to command mode.

a Append Text.

Reads the text entered in input mode and appends it to the buffer after the addressed line. **a** accepts one line-address — default line-address is the current line. The new current line is the last line input, or at the addressed line if no text is entered. Address '0' is a valid place to append text, in which case text is placed at the beginning of the buffer.

c Change Lines.

Deletes the addressed lines, then accepts input text which replaces these lines. **c** accepts two line-addresses — default line-address is the current line. The current line is left on the last line input, or at the line preceding the deleted lines if no text is entered.

d Delete Lines.

Delete the addressed lines from the buffer. **d** accepts two line-addresses — default line-address is the current line. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e *filename* Edit a file.

Deletes the entire contents of the buffer, and then reads in the named file. **e** sets the current line to the last line of the buffer, and reports the number of characters read into the buffer. **e** remembers *filename* for possible use as a default file name in a subsequent **r** or **w** operations. If no *filename* is given, the remembered filename is used. **e** displays a **?**

if the buffer has not been written out since the last change made — a second **e** operation says you really mean it.

E *filename*
Same as **e**, but will silently allow you to quit an editing session without warning you if you have not written your file. **e**, on the other hand, reminds you to save your changes if you have altered the buffer at all.

f *filename* Display Remembered Filename.
Display the currently 'remembered filename'. If *filename* is given, the currently 'remembered filename' is changed to *filename*.

g/*regular expression*/*operation list*
This is the **global** operation: perform *operation list* on all lines in the range of line-addresses containing *regular expression*. **g** accepts two line-addresses — default is all lines in the buffer. Also see the **v** operation, which inverts the sense of *regular expression*.

If your *operation list* actually takes up more than a single line, you must end every line except the last (the true 'end' of the global operation) with an escape character, '\'. For example, if you want to substitute 'jimjams' for 'frammis', then append several lines of text to every line containing the string 'widget' and print those lines, you would type this sequence:

```
g/widget/s/frammis/jimjams/\
a \
new line of text\
another new line of text\
.\
p
```

Note that the **a**, **i**, and **c** operations, which put *ed* in input mode, are permitted in the *operation list*; the final **.** terminating input may be omitted if it is the last line of the *operation list*. The **g** and **v** operations are not permitted in the *operation list*.

i Insert Text.
Insert lines of text into the buffer before the addressed line. **i** accepts one line-address — default line-address is the current line. The current line is placed at the last line input; if no text is input, the current line is left at the line before the addressed line. **i** differs from **a** only in the placement of the text.

j Join Lines.
Joins the addressed lines into a single line; intermediate newlines simply disappear. **j** accepts two line-addresses — default is the current line and the following line. The current line is placed at the resulting line.

k*x* Mark Line.
Marks the addressed line with name *x* (the name must be a lower-case letter). The line-address form **'***x* then addresses this line. **k** accepts one line-address — default line-address is the current line.

l Display Non-printing Characters.
Displays non-graphic characters in the addressed lines such that they are displayed in two-digit octal, and long lines are folded. **l** accepts two line-addresses — default line-address is the current line. **l** may be placed on the same line after any non-I/O operation.

m*address* Move lines.
Reposition the addressed lines after the line-addressed by *address*. **m** accepts two line-addresses to specify the range of lines to be moved — default line-address is the current

line. The last of the moved lines becomes the current line.

p Print (display) Lines.
Displays the addressed lines. **p** accepts two line-addresses — default line-address is the current line. The current line is placed at the last line printed. **p** may be placed on the same line after any non-I/O operation.

P Synonym for **p**.

q Quit Edit Session.
Exit from the editing session. Note, however, that the buffer is not automatically written out (do a '**w**' to write if you want to save your changes). *Ed* warns you once if you haven't saved your file — a second **q** says you really mean it.

Q Same as **q**, but you don't get any warning if you haven't previously written out the buffer.

r *filename* Read from file.
Reads the contents of *filename* into the buffer after the addressed line. If *filename* is not given, the 'remembered filename', if any, is used (see **e** and **f**). **r** accepts one line-address — default line-address is **$**. If line-address '0' is used, **r** reads the file in at the beginning of the buffer. If the read is successful, **r** displays the number of characters read in. The current line is left at the last line read in from the file.

s/*regular expression*/*replacement string*/ or,
s/*regular expression*/*replacement string*/**g**
Substitute the *replacement string* for the first occurrence of *regular expression* on each line where the *regular expression* occurs. In the first form of the **s** operation, only the first occurrence of the matched string on each line is replaced. If you use the **g** (global) suffix, all occurrences of the *regular expression* are replaced in the line. Keep the **g** *suffix* of the **s** operation distinct from the **g** operation itself — they are completely different. **s** accepts two line-addresses to delimit the range of lines within which the substitutions should be done — default line-address is the current line. The current line is left at the last line substituted.

Special Characters:
Any punctuation character may be used instead of '/' to delimit the *regular expression* and the *replacement string*.
An ampersand '&' appearing in the *replacement string* is replaced by the string matching the *regular expression*. The special meaning of '&' in this context may be suppressed by preceding it by '\'.
The characters *n* where *n* is a digit, are replaced by the text matched by the *n*-th *regular subexpression* enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left. Lines may be split by substituting new-line characters into them. The new-line in the *replacement string* must be escaped by preceding it by '\'.

t*address* Transfer Lines.
Transfers a copy of the addressed lines to after line *address*. **t**ransfer is like move, but it makes copies of the lines, leaving the original text where it was. **t** accepts two line-addresses preceeding the operation letter — default line-address is default. The current line is left on the last line of the copy. '0' is a legal line-address for the destination.

u Undo. Undo previous substitute.
undo undoes the effect of the the last substitute operation, *providing that the current line has not been moved since the substitute operation.*

v/regular expression/operation list
Like a negative of the global operation, **g**: perform *operation list* on all lines *except* those containing *regular expression*. **v** accepts two line-addresses — default is all lines in the file.

w Write Lines.

Write the addressed lines from the buffer into the file specified by the 'remembered filename'. **w** accepts two line-addresses — default is all lines in the file. The current line is unchanged. If the write is successful, *ed* displays the number of characters written.

w *filename* Write Lines.
Write the addressed lines into *filename*. *Filename* is created if it does not already exist. *Filename* becomes the 'remembered filename' (see the **e** and **f** operations). **w** accepts two line-addresses — default is all lines in the file. The current line is unchanged. If the write is successful, *ed* displays the number of characters written.

W *filename*
Same as **w**, but appends the addressed lines to the named file instead of overwriting the file. **W** accepts two line-addresses — default is all lines in the file.

x Encrypt File.
When **x** is used, *ed* demands a key string from the standard input. Later **r**, **e**, and **w** operations will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

= Display Line Number.
Display the line number of the addressed line. **=** accepts one line-address — default line-address is **$**. The current line is unchanged by this operation.

!<shell command>
The remainder of the line after the '!' is sent to *sh*(1) to be interpreted as a shell command. The current line is unchanged.

address<**newline**>
Display the addressed line. If you type a line-address and type RETURN, *ed* displays the addressed line. If you simply type RETURN, the line following the current line is displayed (equivalent to '.+1p'). This is useful for stepping through text.

REGULAR EXPRESSIONS

Ed supports a limited form of *regular expression* notation. A regular expression (also known as a *pattern*) specifies a set of strings of characters — such as 'any string containing digits 5 through 9' or 'only lines containing uppercase letters'. A member of this set of strings is said to be *matched* by the regular expression. Regular expressions or patterns are used to address lines in the buffer (see *Line Addressing*, above), and also for selecting strings to be substituted in the **s** (substitute) operation described previously.

An empty regular expression, indicated by two regular expression delimiters in a row, stands for a copy of the last regular expression encountered.

Any given regular expression matches the the longest among the leftmost matches in a line.

In the following specification for regular expressions, the notation *c* stands for any single ordinary character, where a character is anything except a newline character.

c any ordinary character except a special character matches itself. Special characters are the delimiters that actually surround the regular expression, plus \ (the escape character), [(the opening bracket for a character class as described below), . (period which matches any single character), and sometimes the * (closure) ^ and $ characters. If you want a literal occurrence of one of these special characters you must escape them with the \ character.

^ at the very start of the regular expression constrains the match to the beginning of the line. A match of this type is called an 'anchored match' because it is 'anchored' to a specific place in the line. The ^ character loses its special meaning if it appears in any position other than at the very start of the regular expression.

$ at the very end of the regular expression constrains the match to the end of the line. A match of this type is called an 'anchored match' because it is 'anchored' to a specific

place in the line. The $ character loses its special meaning if it appears in any position other than at the very end of the regular expression.

. (period) matches any single character except a newline character.

[*string*] A *string* of characters enclosed in brackets matches any one of the characters in the brackets. For example, [abcxyz] matches any single character from the set 'abcxyz'. If the first character inside the bracket is a ^, the string matches any character **not** inside the brackets. For instance, [^456787] matches any character except '45678'. You can use a shorthand notation to refer to an inclusive *range* of characters: **a–b**. Such a bracketed string of characters is known as a *character class*.

When two regular expressions are concatenated, they match the leftmost and then the longest possible string that can be divided with the first part of the string matching the first regular expression, followed by the second string matching the second regular expression.

* Any regular expression followed by * matches a sequence of 0 or more matches of the regular expression. Such a pattern is called a *closure*. For example: **[a–z][a–z]*** matches any string of one or more lower case letters.

\(*regular expression* \)
The *regular expression* within the \(and \) brackets essentially 'remembers' whatever the *regular expression* matches. This provides a mechanism for extracting parts of strings. There can be up to nine such partial matches in a string. Parenthesized *regular expressions* can be nested.

n where *n* is in the range 1 thru 9, matches a copy of the string that the bracketed regular expression beginning with the *n*th \(matched, as described in the previous paragraph on matching parts of strings. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left.

Regular expressions are used in line-addresses to specify lines and in one operation (see *s* for substitute above) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

FILES

/tmp/e*
ed.hup: work is saved here if telephone hangs up

SEE ALSO

Using the ed Line Editor in *Editing and Text Processing on the Sun Workstation.*

crypt(1)	encode and decode
ex(1)	extended line editor (part of *vi*)
sed(1)	stream editor
vi(1)	visual (display) editor

BUGS

The **l** operation mishandles DEL.
The **undo** operation removes marks from affected lines.
Because 0 is an illegal line-address for a *w* operation, it is not possible to create an empty file with *ed*. Use *cat*(1) to create an empty file.

RESTRICTIONS

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the temporary file. Since each line uses two bytes of memory, the limit on the number of lines should not be exceeded in practice.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. *ed* refuses to read files containing non-ASCII characters.

The encryption facilities of *ed* are not available on software shipped outside the U.S.

NAME

eqn, neqn, checkeq – typeset mathematics

SYNOPSIS

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

DESCRIPTION

Eqn (and *neqn*) are language processors to assist in describing equations. *Eqn* is a preprocessor for *troff*(1) and is intended for devices that can print *troff*'s output. *Neqn* is a preprocessor for *nroff*(1) and is intended for use with terminals. Usage is almost always:

tutorial% **eqn** *file* ... | **troff**
tutorial% **eqn** *file* ... | **nroff**

If no *files* are specified, *eqn* and *neqn* read from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input.

Checkeq reports missing or unbalanced delimiters and .EQ/.EN pairs.

OPTIONS

-d*xy* Set equation delimiters may be set to characters *x* and *y* with the command-line argument. The more common way to do this is with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off' appearing in the text. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

-p*n* Reduce subscripts and superscripts by *n* point sizes from the previous size. In the absence of the **-p** option, subscripts and superscripts are reduced by 3 point sizes from the previous size.

-s*n* Change point size to *n* globally in the document. The point size can also be changed globally in the body of the document by using the **gsize** directive.

-f*n* Change font to *n* globally in the document. The font can also be changed globally in the body of the document by using the **gfont** directive.

EQN LANGUAGE

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

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 + bx+ c}* results in $\frac{1}{\sqrt{ax^2+ bx+ c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n\to\infty} \sum_{0}^{n} x_i$ is made with *lim from {n-> inf} sum from 0 to n x sub i*.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~=~ 1* produces $\left[x^2+ \frac{y^2}{\alpha}\right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for

nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~=~ n under* is $\underline{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \vec{\vec{y}}$.

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

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

Shorthands may be defined or existing keywords redefined with **define**:

define thing % replacement %

defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The *%* may be any character that does not occur in *replacement*.

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

SEE ALSO

troff(1), tbl(1), ms(7), eqnchar(7)
Typesetting Mathematics with 'eqn' and
Formatting Documents with 'nroff' and 'troff'
in *Editing and Text Processing on the Sun Workstation*.

BUGS

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

NAME

error – analyze and disperse compiler error messages

SYNOPSIS

error [**–n**] [**–s**] [**–q**] [**–v**] [**–t** *suffixlist*] [**–I** *ignorefile*] [name]

DESCRIPTION

Error analyzes error messages produced by a number of compilers and language processors. It replaces the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously.

Error looks at error messages, either from the specified file *name* or from the standard input, and:

- Determines which language processor produced each error message,
- Determines the file name and line number of the erroneous line, and,
- Inserts the error message into the source file immediately preceding the erroneous line.

Error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. *Error* touches source files only after all input has been read.

Options are explained later on in this manual entry.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into *error*. For example, when using the *csh* syntax,

tutorial% **make –s lint |& error –q –v**

will analyze all the error messages produced by whatever programs *make* runs when making lint.

Error knows about the error messages produced by: *make*, *cc*, *cpp*, *ccom*, *as*, *ld*, *lint*, *pi*, *pc*, and *f77*. For all languages except Pascal, error messages are restricted to one line. Some error messages refer to more than one line in more than one file, in which case *error* duplicates the error message and inserts it at all of the places referenced.

Error does one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file they are processing. *Error* uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by *error*.

discard
: Error messages from *lint* that refer to one of the two *lint* libraries, */usr/lib/llib-lc* and */usr/lib/llib-port* are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by *error*.

nullify
: Error messages from *lint* can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named *.errorrc* in the user's home directory, or from the file named by the –I option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
: Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They are not inserted into any source file.

file specific
: Error messages that refer to a specific file but to no specific line are written to the standard output when that file is touched.

true errors

Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are inserted into source files. Other error messages are consumed entirely by *error* or are written to the standard output. *Error* inserts the error messages into the source file on the line preceeding the line number in the error message. Each error message is turned into a one line comment for the language, and is internally flagged with the string '###' at the beginning of the error, and '%%%' at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

OPTIONS

–n Do *not* touch any files; all error messages are sent to the standard output.

–q *Error* asks whether the file should be touched. A 'y' or 'n' to the question is necessary to continue. Absence of the –q option implies that all referenced files (except those refering to discarded error messages) are to be touched.

–v After all files have been touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi* can't be found, try *ex* or *ed* from standard places.

–t Take the following argument as a suffix list. Files whose suffices do not appear in the suffix list are not touched. The suffix list is dot seperated, and '*' wildcards work. Thus the suffix list:

".c.y.f*.h"

allows *error* to touch files ending with '.c', '.y', '.f*' and '.y'.

–s Print out *statistics* regarding the error categorization. Not too useful.

Error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

FILES

~/.errorrc	function names to ignore for *lint* error messages
/dev/tty	user's teletype

BUGS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause *error* to not understand the error message.

Error, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the '|' marking the point of error is also disturbed by *error*.

Error was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

NAME

ex, edit – text editor

SYNOPSIS

ex [**–**] [**–v**] [**–t** tag] [**–r**] [**+***command*] [**–l**] name ...
edit [ex options]

DESCRIPTION

Ex is the root of a family of editors which includes *edit*(1), *ex*, and *vi*(1). Display based editing is the focus of *vi*. Most users will in fact use *vi* as the principal interface to the system rather than *ex*.

OPTIONS

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

–v start up in display editing state using *vi*(1). You can achieve the same effect by simply typing the *vi* command itself.

–t *tag* edit the file containing the tag *tag*. A tags database must have been built previously using the *ctags*(1) command.

–r recover the indicated *files* after a crash.

–l set up for editing LISP programs.

–w*nnn* set the default window (number of lines on your terminal) to *nnn*— this is useful if you are dialling into the system over a slow 'phone line.

–x prompt for a key to be used in encrypting the file being edited.

–R edit the file in read-only state. You can achieve the same effect with the *view* command.

+*command*
start the editing session by executing *command*.

DOCUMENTATION

The document *Edit: A tutorial* provides a comprehensive introduction to *edit* assuming no previous knowledge of computers or the UNIX system.

The *Ex Reference Manual – Version 3.5* is a comprehensive and complete manual for the command mode features of *ex*, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed;* the material in the introductory and advanced documents works also with *ex*.

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi*. These documents can be found in the *Editing and Text Processing* Manual. In addition, the *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction*.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), grep(1), vi(1), termcap(5), environ(5)

BUGS

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

Undo never clears the buffer modified condition.

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

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

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

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

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

RESTRICTIONS

The encryption facilities of *ex* are not available on software shipped outside the U.S.

NAME

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

SYNOPSIS

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

DESCRIPTION

Expand copies the named *files* (or the standard input) to the standard output, with tabs changed into spaces (blanks). Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

Unexpand copies the named *files* (or the standard input) to the standard output, putting tabs back into the data. By default only leading spaces (blanks) and tabs are converted to strings of tabs, but this can be overridden by the **–a** option (see the *options* section below).

EXPAND OPTIONS

–tabstop
Specified as a single argument sets tabs *tabstop* spaces apart instead of the default 8.

–tab1,tab2,...,tabn
Set tabs at the columns specified by *tab1...*

UNEXPAND OPTIONS

–a Insert tabs when replacing a run of two or more spaces would produce a smaller output file. This option only applies to *unexpand.*

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

Expr evaluates expressions as specified by its arguments. Each token of the expression is a separate argument. After evaluation, the result is written on the standard output.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

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

expr & *expr*
: yields the first *expr* if neither *expr* is null or '0', otherwise yields '0'.

expr relop expr
: yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expr* are integers, otherwise the comparison is lexicographic. *relop* is one of < (less than), <= (less than or equal to), = (equal to), != (not equal to), >= (greater than or equal to), and > (greater than).

expr + *expr*
expr – *expr*
: addition or subtraction of the arguments.

expr * *expr*
expr / *expr*
expr % *expr*
: multiplication, division, or remainder of the arguments.

expr : *expr*
match *string regular-expression*
: The two forms of the matching operator above are synonymous. The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of *ed*(1). The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

substr *string integer-1 integer-2*
: extracts the subtring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1 has a value greater than string*, *expr* returns a null string. If you try to extract more characters than there are in *string*, *expr* returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as *expr* tends to run forever in these cases.

length *string*
: returns the length (that is, the number of characters) of *string*. index *string character-list* reports the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

(*expr*) parentheses for grouping.

EXAMPLES

To add 1 to the Shell variable *a*:

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

To find the filename part (least significant part) of the pathname stored in variable *a*, which may or may not contain '/':

```
expr $a : '.*/\(.*\)' '|' $a
```

Note the quoted Shell metacharacters.

SEE ALSO

sh(1), test(1)

DIAGNOSTICS

Expr returns the following exit codes:

0	if the expression is neither null nor '0',
1	if the expression is null or '0',
2	for invalid expressions.

BUGS

Note that the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

```
tutorial% expr index expurgatorious length
syntax error
tutorial%
```

generates the 'syntax error' message as shown instead of the value 1 as you might expect.

NAME

eyacc – modified yacc allowing much improved error recovery

SYNOPSIS

eyacc [**–v**] [grammar]

DESCRIPTION

Eyacc is an old version of *yacc*(1), which produces tables used by the Pascal system and its error recovery routines. *Eyacc* fully enumerates test actions in its parser when an error token is in the look-ahead set. This prevents the parser from making undesirable reductions when an error occurs before the error is detected. The table format is different in *eyacc* than it was in the old *yacc*, as minor changes had been made for efficiency reasons.

SEE ALSO

yacc(1)

Practical LR Error Recovery by Susan L. Graham, Charles B. Haley and W. N. Joy; SIGPLAN Conference on Compiler Construction, August 1979.

BUGS

Pc and its error recovery routines should be made into a library of routines for the new *yacc*.

NAME

f77 – FORTRAN-77 compiler

SYNOPSIS

f77 [**-c**] [**-g**] [**-w**] [**-p**] [**-pg**] [**-O**] [**-S**] [**-R**] [**-fsky**] [**-o** *output*] [**-onetrip**] [**-u**] [**-C**] [**-F**] [**-m**] [**-R** *x*] [**-N**[**qxscn**]*nnn*] file ...

DESCRIPTION

F77 is the UNIX FORTRAN-77 compiler for translating programs written in the FORTRAN-77 programming language into executable load modules or relocatable binary programs for subsequent loading via the *ld*(1) linker. In addition to the many flag arguments (options), *f77* accepts several types of files:

Filenames ending in *.f* are taken to be FORTRAN-77 source programs; they are compiled, and each object program is left in the file in the current directory whose name is that of the source with *.o* substituted for *.f*. Filenames ending in *.F* are also taken to be FORTRAN-77 source programs, but they are preprocessed by the C preprocessor (equivalent to a **cc -E**) before they are compiled by the *f77* compiler.

Filenames ending in *.r* are taken to be Ratfor source programs; these are first transformed by the appropriate preprocessor, then compiled by *f77*.

In the same way, filenames ending in *.c* or *.s* are taken to be C or assembly source programs and are compiled or assembled, producing a *.o* file.

OPTIONS

The following options have the same meaning as in *cc*(1). See *ld*(1) for load-time options.

-c Suppress loading and produce *.o* files for each source file.

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

-w Suppress all warning messages. If the option is **-w66**, only FORTRAN-66 compatibility warnings are suppressed.

-D*name*=*def*

-D*name*

Define the *name* to the C preprocessor, as if by '#define'. If no definition is given, the name is defined as "1". (*.F* suffix files only).

-I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **-I** options, then in directories on a standard list. (*.F* suffix files only).

-p Prepare object files for profiling, see *prof*(1).

-pg Produce counting code in the manner of **-p**, but invoke a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. An execution profile can then be generated by use of *gprof*(1).

-O Optimize the object-code.

-S Compile the named programs, and leave the assembler-language output on corresponding files suffixed *.s* (no *.o* file is created).

-o *output*

Name the final output file *output* instead of *a.out*.

-fsky Generate code which assumes the presence of a SKY floating-point processor board. Programs compiled with this option can only be run in systems that have a SKY board installed. Programs compiled without the **-fsky** option will use the SKY board, but won't run as fast as they would if the **-fsky** option were used. If any part of a program is compiled using the **-fsky** option, you must also use this option when linking with the *f77* command, since a different set of startup routines is used.

The following options are peculiar to *f77*:

-i2 On machines which support short integers, make the default integer constants and variables short. (**-i4** is the standard value of this option). All logical quantities will be short.

-m Apply the M4 preprocessor to each *.r* file before transforming it with the Ratfor preprocessor.

-onetrip
Compile DO loops that are performed at least once if reached. FORTRAN-77 DO loops are not performed at all if the upper limit is smaller than the lower limit.

-u Make the default type of a variable 'undefined' rather than using the default FORTRAN rules.

-v Print the version number of the compiler, and the name of each pass as it executes.

-C Compile code to check that subscripts are within declared array bounds.

-F Apply the C or Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to *.f*, but do not compile.

-R*x* Use the string *x* as a Ratfor option in processing *.r* files.

-N[qxscn]*nnn*
Make static tables in the compiler bigger. *F77* complains if tables overflow and suggests you apply one or more of these flags. These flags have the following meanings:

q Maximum number of equivalenced variables. Default is 150.

x Maximum number of external names (common block names, subroutine and function names). Default is 200.

s Maximum number of statement numbers. Default is 401.

c Maximum depth of nesting for control statements (for example, DO loops). Default is 20.

n Maximum number of identifiers. Default is 1009.

-U Do not convert upper case letters to lower case. The default is to convert FORTRAN programs to lower case except within character string constants.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program called *a.out*.

FILES

file.[fFrsc]	input file
file.o	object file
a.out	loaded output
./fort[pid].?	temporary
/usr/lib/f77pass1	compiler
/lib/f1	pass 2
/lib/c2	optional optimizer
/lib/cpp	C preprocessor
/usr/lib/libf77.a	Fortran library
/lib/libc.a	C library, see section 3
mon.out	file produced for analysis by prof(1).
gmon.out	file produced for analysis by gprof(1).

SEE ALSO

FORTRAN Programmer's Guide for the Sun Workstation.
prof(1), gprof(1), cc(1), ld(1), ratfor(1)

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

BUGS

The FORTRAN-66 subset of the language has been exercised extensively; the newer features have not.

NAME

false, true – provide truth values

SYNOPSIS

true

false

DESCRIPTION

True and *false* are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

EXAMPLE

```
while false
do
    command list
done
```

SEE ALSO

csh(1), sh(1), true(1)

DIAGNOSTICS

False has exit status nonzero.

NAME

file – determine file type

SYNOPSIS

file file ...

DESCRIPTION

File performs a series of tests on each *file* in an attempt to determine what its contents are. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language.

EXAMPLE

The example illustrates the use of *file* on all the files in a specific user's directory:

```
% pwd
/usr/henry/misc
% ls
bensusan        fortran.mss     messages        romero          toolkit.tr
command.list    jokes           pascal.mss      squash          useful.news
counts          letters         play.mss        tech.papers     v7.stuff
deuter          memos           roadmap.mss     titles          window
% file *
bensusan:       roff, nroff, or eqn input text
command.list:   roff, nroff, or eqn input text
counts:         ascii text
deuter:         roff, nroff, or eqn input text
fortran.mss:    roff, nroff, or eqn input text
jokes:          directory
letters:        directory
memos:          directory
messages:       directory
pascal.mss:     roff, nroff, or eqn input text
play.mss:       roff, nroff, or eqn input text
roadmap.mss:    roff, nroff, or eqn input text
romero:         roff, nroff, or eqn input text
squash:         directory
tech.papers:    directory
titles:         ascii text
toolkit.tr:     roff, nroff, or eqn input text
useful.news:    directory
v7.stuff:       archive
window:         directory
%
```

BUGS

It often makes mistakes. In particular it often suggests that command files are C programs.

Does not recognize Pascal or LISP.

NAME

find – find files

SYNOPSIS

find pathname-list expression

DESCRIPTION

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

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

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

–type c
: True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **f** or **l** for block special file, character special file, directory, plain file, or symbolic link.

–links n
: True if the file has *n* links.

–user uname
: True if the file belongs to the user *uname* (login name or numeric user ID).

–group gname
: True if the file belongs to group *gname* (group name or numeric group ID).

–size n
: True if the file is *n* blocks long (512 bytes per block).

–inum n
: True if the file has inode number *n*.

–atime n
: True if the file has been accessed in *n* days.

–mtime n
: True if the file has been modified in *n* days.

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

–ok command
: Like **–exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

–print
: Always true; the current pathname is printed.

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

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

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).
2) The negation of a primary ('!' is the unary *not* operator).
3) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
4) Alternation of primaries ('**–o**' is the *or* operator).

EXAMPLE

In our local development system, we keep a file called *TIMESTAMP* in all the manual page directories. Here is how to find all entries that have been updated since *TIMESTAMP* was created:

```
angel% find /usr/man/man2 -newer /usr/man/man2/TIMESTAMP -print
/usr/man/man2
/usr/man/man2/socket.2
/usr/man/man2/mmap.2
angel%
```

To find all the files called *intro.mss* starting from the current user's directory:

```
angel% find . -name intro.mss -print
./manuals/assembler/intro.mss
./manuals/sun.core/intro.mss
./manuals/driver.tut/intro.mss
./manuals/users.guide/intro.mss
./manuals/refman/intro.mss
./manuals/sys.manager/intro.mss
./manuals/sys.manager/uucp.impl/intro.mss
./supplements/general.works/unix.introduction/intro.mss
./supplements/programming.tools/sccs/intro.mss
angel%
```

To remove all files named 'a.out' or '*.o' that have not been accessed for a week:

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

Note that the { } characters must be quoted when using the C shell.

FILES

/etc/passwd
/etc/group

SEE ALSO

sh(1), test(1), fs(5)

BUGS

The syntax is painful.

NAME

fmt – simple text formatter

SYNOPSIS

fmt [file ...]

DESCRIPTION

Fmt is a simple text formatter which reads the concatenation of input files (or standard input if none are given) and produces on standard output a version of its input with lines as close to 72 characters long as possible. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

Fmt is meant to format mail messages prior to sending, but may also be useful for other simple tasks. For instance, in the *vi* text editor, the command:

```
!}fmt
```

reformats a paragraph, making the lines reasonably even length.

SEE ALSO

nroff(1), mail(1)

BUGS

Fmt was designed to be simple and fast – for more complex operations, the standard text processors are likely to be more appropriate.

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [*–width*] [file ...]

DESCRIPTION

Fold is a filter which folds the contents of the specified *files*, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1) before using *fold*.

SEE ALSO

expand(1)

BUGS

Folding may not work correctly if underlining is present.

NAME

fpr – print Fortran file

SYNOPSIS

fpr

DESCRIPTION

Fpr is a filter that transforms files formatted according to Fortran's carriage control conventions into files formatted according to UNIX line printer conventions.

Fpr copies its input onto its output, replacing the carriage control characters with characters that will produce the intended effects when printed using *lpr*(1). The first character of each line determines the vertical spacing as follows:

(blank) one line

0 two lines

1 to first line of next page

+ no advance

A blank line (that is, an empty line) is treated as if its first character is a blank. A blank that appears as a carriage control character is deleted. A zero is changed to a newline. A one is changed to a form feed. The effects of a "+" are simulated using backspaces.

Note that *fpr* is known as *asa* in UNIX System V.

EXAMPLES

```
a.out | fpr | lpr

fpr < f77.output | lpr
```

BUGS

Results are undefined for input lines longer than 170 characters.

NAME

from – who is my mail from?

SYNOPSIS

from [**–s***sender*] [user]

DESCRIPTION

From prints out the mail header lines in your mailbox file to show you who your mail is from. If *user* is specified, then *user*'s mailbox is examined instead of your own.

OPTIONS

–s*sender*

Only display headers for mail sent by *sender.*

FILES

/usr/spool/mail/*

SEE ALSO

biff(1), mail(1), prmail(1)

NAME

fsplit – split a multi-routine Fortran file into individual files

SYNOPSIS

fsplit [–**e** efile] ... [file]

DESCRIPTION

Fsplit takes as input either a file or standard input containing Fortran source code. It attempts to split the input into separate routine files of the form *name.f*, where *name* is the name of the program unit (e.g. function, subroutine, block data or program). The name for unnamed block data subprograms has the form *blkdtaNNN.f* where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form *mainNNN.f*. If there is an error in classifying a program unit, or if *name.f* already exists, the program unit will be put in a file of the form *zzzNNN.f* where *zzzNNN.f* does not already exist.

Normally each subprogram unit is split into a separate file. When the *–e* option is used, only the specified subprogram units are split into separate files. E.g.:

```
fsplit –e readit –e doit prog.f
```

will split readit and doit into separate files.

DIAGNOSTICS

If names specified via the *-e* option are not found, a diagnostic is written to *standard error*.

BUGS

Fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse *fsplit*.

It is hard to use –**e** for unnamed main programs and block data subprograms since you must predict the created file name.

NAME

ftp – file transfer program

SYNOPSIS

ftp [**–v**] [**–d**] [**–i**] [**–n**] [*host*]

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. *Ftp* transfers files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* immediately attempts to establish a connection to an FTP server on that host; otherwise, *ftp* enters its command interpreter and awaits instructions from the user. When *ftp* is awaiting commands from the user, it displays the prompt "ftp>". *Ftp* recognizes the following commands:

! Invoke a shell on the local machine.

ascii Set the file transfer *type* to network ASCII. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the file transfer *type* to support binary image transfer.

bye Terminate the FTP session with the remote server and exit *ftp*.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close Terminate the FTP session with the remote server, and return to the command interpreter.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string "–>".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal.

form *format*
Set the file transfer *form* to *format*. The default format is "file".

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file

is specified, the output is sent to the terminal.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

mkdir *directory-name*
Make a directory on the remote machine.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any mget or mput will transfer all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for bye.

quote*arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

recv *remote-file* [*local-file*]
A synonym for get.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for put.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default "stream" structure is used.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it.

Unless *ftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for help.

Command arguments which have imbedded spaces may be quoted with quote (") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name "" is specifed, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is "|", the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell, using *popen*(3S) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ""| ls -lt"". A particularly useful example of this mechanism is: "dir |more".

3) Failing the above checks, local file names are expanded according to the rules used in the *csh*(1). (This is not enabled as multiple file transfers are currently not supported.)

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of "ascii", "image" (binary), "ebcdic", and "local byte size" (for PDP-10's and PDP-20's mostly). *Ftp* supports the ascii and image types of file transfer.

Ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

-v show all responses from the remote server, as well as report on data transfer statistics.

-n do not attempt "auto-login" upon initial connection. If auto-login is enabled, *ftp* checks the *.netrc* file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* uses the login name on the local machine as the user identity on the remote machine, and prompts for a password and, optionally, an account with which to login.

-i turn on interactive prompting during mutliple file transfers (unimplemented).

-d enable debugging.

BUGS

Many FTP server implementations do not support experimental operations such as print working directory. Aborting a file transfer does not work right; if one attempts this the local *ftp* will likely have to be killed by hand. VAX sites running the BBN FTP server appear to ignore the PORT command while indicating complicity; this locks up all file transfers.

NAME

gcore – get core images of running processes

SYNOPSIS

gcore process-id ...

DESCRIPTION

Gcore creates a core image of each specified process. Such an image can be used with *adb*(1) or *dbx*(1).

FILES

core.<process-id> core images

BUGS

Paging activity that occurs while *gcore* is running may cause the program to become confused. For best results, the desired processes should be stopped.

NAME

get – get a version of an SCCS file

SYNOPSIS

/usr/sccs/get [**–r SID**] [**–c cutoff**] [**–i list**] [**–x list**] [**–a seq-no.**] [**–k**] [**–e**] [**–l**[**p**]] [**–p**]
[**–m**] [**–n**] [**–s**] [**–b**] [**–g**] [**–t**] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specified option. Arguments may be specified in any order, options apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

OPTIONS

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

–r *SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the **–e** option is also used), as a function of the SID specified.

–c *cutoff*

Cutoff date-time, in the form: YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **–c7502** is equivalent to **–c750228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: **–c77/2/2 9:22:25**. Note that this implies that one may use the %E% and %U% identification keywords.

–e This *get* is for editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). A **/usr/sccs/get –e** applied to a particular version (SID) of the SCCS file prevents further **/usr/sccs/get –e** commands on the same SID until *delta* is run or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **/usr/sccs/get –e** for different SIDs is always allowed.

If the *g-file* generated by a **/usr/sccs/get –e** is accidentally ruined in the process of editing it, it may be regenerated by re-running a *get* with the **–k** option in place of the **–e** option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **–e** option is used.

–b Used with the **–e** option to indicate that the new delta should have an SID in a new branch as shown in Table 1. This option is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

–i *list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the 'SID Specified' column of Table 1. Partial SIDs are interpreted as shown in the 'SID Retrieved' column of Table 1.

-x *list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **-i** option for the *list* format.

-k Suppress replacement of identification keywords (see below) in the retrieved text by their value. The **-k** option is implied by the **-e** option.

-l[p] Write a delta summary into an *l-file*. If **-lp** is used, the delta summary is written on the standard output and the *l-file* is not created. See *FILES* for the format of the *l-file*.

-p Write the text retrieved from the SCCS file to the standard output. No *g-file* is created. All output which normally goes to the standard output goes to the standard error file instead, unless the **-s** option is used, in which case it disappears.

-s Suppress all output normally written on the standard output. However, fatal error messages (which always go to the standard error file) remain unaffected.

-m Precede each text line retrieved from the SCCS file with the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n Precede each generated text line with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** options are used, the format is: %M% value, followed by a horizontal tab, followed by the **-m** option generated format.

-g Do not actually retrieve text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

-t Access the most recently created ('top') delta in a given release (for example, **-r1**), or release and level (for example, **-r1.2**).

-a *seq-no.*
The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This option is used by the *comb*(1) command; it is not a generally useful option, and users should not use it. If both the **-r** and **-a** options are specified, the **-a** option is used. Care should be taken when using the **-a** option in conjunction with the **-e** option, as the SID of the delta to be created may not be what one expects. The **-r** option can be used with the **-a** and **-e** options to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** option is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** option is used included deltas are listed following the notation 'Included'; if the **-x** option is used, excluded deltas are listed following the notation 'Excluded'.

TABLE 1. Determination of SCCS Identification String

SID* Specified	–b Option Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* 'R', 'L', 'B', and 'S' are the 'release', 'level', 'branch', and 'sequence' components of the SID, respectively; 'm' means 'maximum'. Thus, for example, 'R.mL' means 'the maximum level number within release R'; 'R.L.(mB+1).1' means 'the first sequence number on the *new* branch (that is, maximum branch number plus one) of level L within release R'. Note that if the SID specified is of the form 'R.L', 'R.L.B', or 'R.L.B.S', each of the specified components *must* exist.

** 'hR' is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** Forces creation of the *first* delta in a *new* release.

\# Successor.

† The –b option is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of – means 'irrelevant'.

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as 'this shouldn't have happened' type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*, These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **–p** option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **–k** option is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **–l** option is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   * if the delta wasn't applied and wasn't ignored.
c. A code indicating a 'special' reason why the delta was or was not applied:
   'I': Included.
   'X': Excluded.
   'C': Cut off (by a **–c** option).
d. Blank.
e. SCCS identification (SID).
f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created *delta*.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* passes information resulting from a **/usr/sccs/get** **-e** along to *delta*. Its contents are also used to prevent a subsequent execution of a **/usr/sccs/get** **-e** for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **-i** option if it was present, followed by a blank and the **-x** option if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (that is, *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, only one file may be named when the **-e** option is used.

NAME

gprof – display call graph profile data

SYNOPSIS

gprof [**–a**] [**–b**] [**–c**] [**–e** *name*] [**–E** *name*] [**–f** *name*] [**–F** *name*] [**–s**] [**–z**]
[a.out [gmon.out ...]]

DESCRIPTION

gprof produces an execution profile of C, Pascal, or Fortran77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (*gmon.out* default) which is created by programs compiled with the **–pg** option of *cc*, *pc*, and *f77*. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the named object file (*a.out* default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendents. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendents is propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

OPTIONS

–a suppresses the printing of statically declared functions. If this option is given, all relevant information about the static function (for instance, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

–b display a description of each field in the profile.

–c the static call graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

–e *name*
suppresses the printing of the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **–e** option may be given. Only one *name* may be given with each **–e** option.

–E *name*
suppresses the printing of the graph profile entry for routine *name* (and its descendants) as **–e**, above, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **–E** *mcount* **–E** *mcleanup* is the default.)

–f *name*
prints the graph profile entry of only the specified routine *name* and its descendants. More than one **–f** option may be given. Only one *name* may be given with each **–f** option.

–F *name*
prints the graph profile entry of only the routine *name* and its descendants (as **–f**, above) and also uses only the times of the printed routines in total time and percentage computations. More than one **–F** option may be given. Only one *name* may be given with each

-F option. The **-F** option overrides the **-E** option.

-s a profile file *gmon.sum* is produced which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of gprof (probably also with a **-s**) to accumulate profile data across several runs of an *a.out* file.

-z displays routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **-c** option for discovering which routines were never called.

FILES

a.out	the namelist and text space.
gmon.out	dynamic call graph and profile.
gmon.sum	summarized dynamic call graph and profile.

SEE ALSO

monitor(3), profil(2), cc(1), prof(1)
Graham, S.L., Kessler, P.B., McKusick, M.K.,
"gprof: A Call Graph Execution Profiler",
Proceedings of the SIGPLAN '82 Symposium on Compiler Construction,
SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. We assume that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call *exit*(2) or return normally for the profiling information to be saved in the gmon.out file.

NAME

graph – draw a graph

SYNOPSIS

graph [**-a** *spacing* [*start*]] [**-b**] [**-c** *string*] [**-g** *gridstyle*] [**-l** *label*]
[**-m** *connectmode*] [**-s**] [**-x** [**l**] *lower* [*upper* [*spacing*]]]
[**-y** [**l**] *lower* [*upper* [*spacing*]]] [**-h** *fraction*] [**-w** *fraction*] [**-r** *fraction*]
[**-u** *fraction*] [**-t**] ...

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *plot*(1G) filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "...", in which case they may be empty or contain blanks and numbers; labels never contain newlines.

A legend indicating grid range is produced with a grid unless the **-s** option is present.

OPTIONS

Each option is recognized as a separate argument.

-a *spacing* [*start*]
: Supply abscissas automatically (they are missing from the input); *spacing* is the spacing (default 1). *start* is the starting point for automatic abscissas (default 0 or lower limit given by **-x**).

-b
: Break (disconnect) the graph after each label in the input.

-c *string*
: *String* is the default label for each point.

-g *gridstyle*
: *Gridstyle* is the grid style: 0 no grid, 1 frame with ticks, 2 full grid (default).

-l
: *label* is label for graph.

-m *connectmode*
: is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

-s
: Save screen, don't erase before plotting.

-x [**l**] *lower* [*upper* [*spacing*]]
: If **l** is present, *x* axis is logarithmic. *lower* and *upper* are lower (and upper) *x* limits. *spacing*, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

-y [**l**] *lower* [*upper* [*spacing*]]
: If **l** is present, *y* axis is logarithmic. *lower* and *upper* are lower (and upper) *y* limits. *spacing*, if present, is grid spacing on *y* axis. Normally these quantities are determined automatically.

-h *fraction*
: is fraction of space for height.

-w *fraction*
: is fraction of space for width.

-r *fraction*
: is fraction of space to move right before plotting.

-u *fraction*
: is fraction of space to move up before plotting.

-t Transpose horizontal and vertical axes. (Option **-x** now applies to the vertical axis.)

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1G), plot(1G)

BUGS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

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

SYNOPSIS

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

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

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

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular expressions in the style of *ex*(1). *Egrep* patterns are full regular expressions including alternation. *Fgrep* searches for lines that contain one of the (newline-separated) *strings*. *Fgrep* patterns are fixed strings — no regular expression metacharacters are supported.

In general, *egrep* is the fastest of these programs.

Take care when using the characters **$ * [^ | ( )** and **** in the *expression* as these characters are also meaningful to the Shell. Enclose the entire *expression* argument in single quotes **' '** if you need any of the above special characters in the *expression*.

When any of the *grep* utilities is applied to more than one input file, the name of the file is displayed preceding each line which matches the pattern. The filename is not displayed when processing a single file, so if you actually want the filename to appear, use */dev/null* as a second file in the list.

OPTIONS

–v Invert the search to only display lines that *do not* match.

–x Display only those lines which match exactly — that is, only lines which match in their entirety (*fgrep* only).

–c Display a count of matching lines.

–l List the names of files with matching lines (once) separated by newlines.

–n Precede each line by its relative line number in the file.

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

–i Ignore the case of letters in making comparisons — that is, upper and lower case are considered identical. This applies to *grep* and *fgrep* only.

–s Work silently, that is, display nothing except error messages. This is useful for checking the error status.

–w search for the expression as a word as if surrounded by '\<' and '\>', see *ex*(1). *grep* only.

–e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a –.

–f *file* Take the regular expression (*egrep*) or string list (*fgrep*) from *file*.

REGULAR EXPRESSIONS

In the following description 'character' excludes newline:

**** Is an escape character: \ followed by any single character other than newline matches that character.

^ Anchored match: matches the beginning of a line.

$ Anchored match: matches the end of a line.

. (period) matches any character.

c Where *c* is any single character not otherwise endowed with special meaning matches that character.

[*string*]
Character class: match any single character from *string*. Ranges of ASCII character codes may be abbreviated as in 'a-z0-9'. A] may occur only as the first character of the string. A literal - must be placed where it can't be mistaken as a range indicator. A ^ (circumflex) character immediately after the open bracket negates the sense of the character class, that is, the pattern matches any character *except* those in the character class.

* Closure: a regular expression followed by an * (asterisk) matches a sequence of 0 or more matches of the regular expression.

\+ Closure: a regular expression followed by a + (plus) matches a sequence of 1 or more matches of the regular expression.

? Closure: a regular expression followed by a ? (question mark) matches a sequence of 0 or 1 matches of the regular expression.

concatenation
Two regular expressions concatenated match a match of the first followed by a match of the second.

| Alternation: two regular expressions separated by | or newline match either a match for the first or a match for the second (*egrep* only).

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

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

EXAMPLES

Search a file for a fixed string using *fgrep*:

```
% fgrep intro /usr/man/man3/*.3*
```

Look for character classes using *grep*:

```
% grep '[1-8]([CJMSNX])' /usr/man/man1/*.1
```

Look for alternative patterns using *egrep*:

```
% 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:

```
% grep 'Sally Parker' telephone.list /dev/null
```

SEE ALSO

ex(1), sed(1), sh(1)

DIAGNOSTICS

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

BUGS

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

Ideally there should be only one *grep*, but for historical reasons there are three different versions each with a slightly different set of options and syntaxes.

NAME

groups – show group memberships

SYNOPSIS

groups

DESCRIPTION

Groups displays the groups to which you belong. Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group*. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

When a new file is created it is given the group of the containing directory.

SEE ALSO

setgroups(2)

FILES

/etc/passwd, /etc/group

NAME

head – display first few lines of specified files

SYNOPSIS

head [–count] [file ...]

DESCRIPTION

Head copies the first *count* lines of the specified file(s), or of the standard input if no filename is given, to the standard output. The default value of *count* is 10 lines.

When more than one file is specified, *head* places a marker at the start of each file which looks like:

==> *filename* <==

Thus, a common way to display a set of short files, identifying each one, is:

gaia% **head -9999 file1 file2 ...**

EXAMPLE

```
gaia% head -4 /usr/man/man1/{cat,head,tail}.1
==> /usr/man/man1/cat.1 <==
.TH CAT 1 "2 June 1983"
.SH NAME
cat – concatenate and display
.SH SYNOPSIS

==> /usr/man/man1/head.1 <==
.TH HEAD 1 "24 August 1983"
.SH NAME
head – display first few lines of specified files
.SH SYNOPSIS

==> /usr/man/man1/tail.1 <==
.TH TAIL 1 "27 April 1983"
.SH NAME
tail – display the last part of a file
.SH SYNOPSIS
```

SEE ALSO

more(1), tail(1), cat(1)

NAME

help – ask for help

SYNOPSIS

/usr/sccs/help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1	Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **ge6**, for message 6 from the *get* command).
type 2	Does not contain numerics (as a command, such as **get**)
type 3	Is all numeric (for example, **212**)

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

When all else fails, try **/usr/sccs/help stuck**.

FILES

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

hostid – print identifier of current host system

SYNOPSIS

hostid

DESCRIPTION

The *hostid* command prints the identifier of the current host in hex. This numeric value is unique across all hosts.

SEE ALSO

gethostid(2)

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

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

SEE ALSO

gethostname(2), sethostname(2)

NAME

indent – indent and format C program source

SYNOPSIS

indent *input* [*output*] [*flags*]

DESCRIPTION

Indent is intended primarily as a C program formatter. Specifically, *indent* will:

- indent code lines
- align comments
- insert spaces around operators where necessary
- break up declaration lists as in "int a,b,c;".

Indent will not break up long statements to make them fit within the maximum line length, but it will flag lines that are too long. Lines will be broken so that each statement starts a new line, and braces will appear alone on a line. (See the –br option to inhibit this.) Also, an attempt is made to line up identifiers in declarations.

The *flags* which can be specified follow. They may appear before or after the file names. If the *output* file is omitted, the formatted file will be written back into *input* and a "backup" copy of *input* will be written in the current directory. If *input* is named "/blah/blah/file", the backup file will be named ".Bfile". If *output* is specified, *indent* checks to make sure it is different from *input*.

The following flags may be used to control the formatting style imposed by *indent*.

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

–c*nnn* The column in which comments will start. The default is 33.

–cd*nnn* The column in which comments on declarations will start. The default is for these comments to start in the same column as other comments.

–l*nnn* The number of spaces for one indentation level. The default is 4.

–dj,–ndj **–dj** will cause declarations to be left justified. **–ndj** will cause them to be indented the same as code. The default is **–ndj**.

–v,–nv **–v turns on "verbose" mode, –nv** turns it off. When in verbose mode, *indent* will report when it splits one line of input into two or more lines of output, and it will give some size statistics at completion. The default is **–nv**.

–bc,–nbc If **–bc** is specified, then a newline will be forced after each comma in a declaration. **–nbc** will turn off this option. The default is **–bc**.

–d*nnn* This option controls the placement of comments which are not to the right of code. Specifying **–d2** means that such comments will be placed two indentation levels to the left of code. The default **–d0** lines up these comments with the code. See the section on comment indentation below.

–br,–bl Specifying **–bl** will cause complex statements to be lined up like this:

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

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

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

You may set up your own "profile" of defaults to *indent* by creating the file ".indent.pro" in your login directory and including whatever switches you like. If *indent* is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, will always override profile switches. The profile file must be a single line of not more than 127 characters. The switches should be separated on the line by spaces or tabs.

Multi-line expressions

Indent will not break up complicated expressions that extend over multiple lines, but it will usually correctly indent such expressions which have already been broken up. Such an expression might end up looking like this:

```
x =
        (
            (Arbitrary parenthesized expression)
            +
            (
                (Parenthesized expression)
                *
                (Parenthesized expression)
            )
        );
```

Comments

Indent recognizes four kinds of comments. They are: straight text, "box" comments, UNIX-style comments, and comments that should be passed through unchanged. The action taken with these various types are as follows:

"Box" comments. *Indent* assumes that any comment with a dash immediately after the start of comment (i.e. "/*-") is a comment surrounded by a box of stars. Each line of such a comment will be left unchanged, except that the first non-blank character of each successive line will be lined up with the beginning slash of the first line. Box comments will be indented (see below).

"Unix-style" comments. This is the type of section header which is used extensively in the UNIX system source. If the start of comment ("/*") appears on a line by itself, indent assumes that it is a UNIX-style comment. These will be treated similarly to box comments, except the first non-blank character on each line will be lined up with the "*" of the "/*".

Unchanged comments. Any comment which starts in column 1 will be left completely unchanged. This is intended primarily for documentation header pages. The check for unchanged comments is made before the check for UNIX-style comments.

Straight text. All other comments are treated as straight text. *Indent* will fit as many words (separated by blanks, tabs, or newlines) on a line as possible. Straight text comments will be indented.

Comment indentation

Box, UNIX-style, and straight text comments may be indented. If a comment is on a line with code it will be started in the "comment column", which is set by the **-cnnn** command line parameter. Otherwise, the comment will be started at *nnn* indentation levels less than where code is currently being placed, where *nnn* is specified by the **-dnnn** command line parameter. (Indented comments will never be placed in column 1.) If the code on a line extends past the comment column, the comment will be moved to the next line.

DIAGNOSTICS

Diagnostic error messages, mostly to tell that a text line has been broken or is too long for the output line.

FILES

.indent.pro profile file

BUGS

Does not know how to format "long" declarations.

NAME

indxbib – make inverted index to a bibliography
lookbib – find references in a bibliography

SYNOPSIS

indxbib database ...
lookbib database

DESCRIPTION

Indxbib makes an inverted index to the named *databases* (or files) for use by *lookbib*(1) and *refer*(1). These files contain bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with "%".

Indxbib is a shell script that calls /usr/lib/refer/mkey and /usr/lib/refer/inv. The first program, *mkey*, truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed; see page 4 of the *Refer* document by Mike Lesk. The second program, *inv*, creates an entry file (.ia), a posting file (.ib), and a tag file (.ic), all in the working directory.

Lookbib uses an inverted index made by *indxbib* to find sets of bibliographic references. It reads keywords typed after the ">" prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another ">" prompt.

It is possible to search multiple databases, as long as they have a common index made by *indxbib*. In that case, only the first argument given to *indxbib* is specified to *lookbib*.

If *lookbib* does not find the index files (the .i[abc] files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a '.ig' suffix, suitable for use with *fgrep*. It then uses this fgrep file to find references. This method is simpler to use, but the .ig file is slower to use than the .i[abc] files, and does not allow the use of multiple reference files.

FILES

x.ia, *x*.ib, *x*.ic, where *x* is the first argument, or if these are not present, then *x*.ig, *x*

SEE ALSO

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

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

inews – submit news articles

SYNOPSIS

inews [**–h**] **–t** *title* [**–n** *newsgroups*] [**–e** *expiration date*]

inews –p [*filename*]

inews –C *newsgroup*

DESCRIPTION

Inews submits news articles to the USENET news network. *Inews* is intended as a raw interface to the news system, not as a human user interface. Casual users should probably use *postnews*(1) instead.

The first form of *inews* is for submitting user articles. The body of the article is read from the standard input. A *title* must be specified as there is no default. Each article belongs to a list of newsgroups. The standard list of newsgroups is used if the list is not specified via the –n option. On the Sun system, the standard subscription list is: *general, all.general, general, all.announce, !junk, !control*, and *!test*. If you wish to submit an article in multiple newsgroups, the *newsgroups* must be separated by commas and/or spaces. The expiration date is set to the local default if not otherwise specified.

When posting an article, the environment is checked for information about the sender. If NAME is found, its value is used for the full name, rather than the system value obtained from */etc/passwd*. This is useful if the system value cannot be set, or when more than one person uses the same login. If ORGANIZATION is found, the value overrides the system default organization. This is useful when a person uses a guest login and is not primarily associated with the organization owning the machine.

The second form of *inews* is for receiving articles from other machines. If *filename* is given, the article is read from the specified file; otherwise the article is read from the standard input. An expiration date need not be present and a receival date is ignored if present.

After local installation, *inews* transmits the article to all systems that subscribe to the newsgroups that the article belongs to.

The third form of *inews* is for creating new newsgroups. On some systems, this may be limited to specific users such as the super-user or news administrator. This is true on the Sun system.

If the file */usr/lib/news/recording* is present, it is taken as a list of 'recordings' to be shown to users posting news. This is an analogy to the recording you hear when you dial information in some parts of the country, asking you if you really wanted to do this. */usr/lib/news/recording* contains lines of the form:

newsgroups <tab> filename

for example:

net.all net.recording fa.all fa.recording

Any user posting an article to a newsgroup matching the pattern on the left is shown the contents of the file on the right. The file is found in the LIB directory (often */usr/lib/news*). The user is then told to hit DEL to abort or RETURN to proceed. The intent of this feature is to help companies keep proprietary information from accidently leaking out.

OPTIONS

–n " *newsgroups*"

specifies a list of newsgroups to which the articles are submitted. Elements in the list must be separated by commas and/or spaces. The expiration date is set to the local default if not otherwise specified.

–f [*sender*]

Specifies the article's sender. Without this flag, the sender defaults to the user's name. If **–f** is specified, the real sender's name is included as a Sender line.

-h Headers are present at the beginning of the article, and these headers should be included with the article header instead of as text. This mechanism can be used to edit headers and supply additional nondefault headers, but not to specify certain information, such as the sender and article ID that *inews* itself generates.

FILES

/usr/spool/news/.sys.nnn	temporary articles
/usr/spool/news/*newsgroups/article_no.*	Articles
/usr/spool/oldnews/	Expired articles
/usr/lib/news/active	List of known newsgroups and highest local article numbers in each.
/usr/lib/news/seq	Sequence number of last article
/usr/lib/news/history	List of all articles ever seen
/usr/lib/news/sys	System subscription list

SEE ALSO

mail(1), binmail(1), getdate(3), news(5), newsrc(5), postnews(1), readnews(1), recnews(1), sendnews(8), uucp(1), uurec(8),
Network News User's Guide in the *Beginner's Guide to the Sun Workstation.*

NAME

install – install files

SYNOPSIS

install [**-c**] [**-m** *mode*] [**-o** *owner*] [**-g** *group*] [**-s**] binary destination

DESCRIPTION

Binary is copied to *destination*. If *destination* already exists, it is removed before *binary* is copied. If the destination is a directory then *binary* is copied into file *destination/binary*.

Install refuses to move a file onto itself.

Note: *install* has no special privileges since it simply uses *cp* to copy files from one place to another. The implications of this are:

- You must have permission to read *binary*.
- You must have permission to copy into *destination*.
- You must have permission to change the modes on the final copy of the file if you want to use the **-m** option to change modes. In addition, if you want to set any modes (such as set-user-id), you must be super-user.
- You must be super-user if you want to use the **-o** option to change ownership.

OPTIONS

-c Copy *binary* instead of moving it. In fact, *install always copies the file, but the* **-c** option is retained for backwards compatibility with old system shell scripts which might otherwise break.

-m *mode*
Specifies a different mode for *binary*: the mode for *destination* is set to 755 by default.

-o *owner*
Set the owner of the *destination* file to *owner*. *destination* is changed to owner *root* by default.

-g *group*
Set the group ownership of the *destination* file to *group*. *destination* is changed to group *staff* by default.

-s Strip binary files after it is installed — only applicable to binary files in *a.out*(5) format.

SEE ALSO

chmod(1), cp(1), mv(1), strip(1), chown(8)

BUGS

Should be possible to move multiple files at a time, like *mv*(1) or *cp*(1).

When the destination is a directory, *install* simply appends the entire source file name to the directory name, instead of using the source file name's last component like *mv*(1) or *cp*(1).

NAME

join – relational database operator

SYNOPSIS

join [**-a***n*] [**-e***string*] [**-j***n.m*] [**-o** *list*] [**-t***c*] file1 file2

DESCRIPTION

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

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

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

Fields are separated by blanks, tabs or newlines. Multiple separators count as one, and leading separators are discarded.

OPTIONS

-a*n* The parameter *n* can be one of the values:

1 produce a line for each unpairable line in *file1*.

2 produce a line for each unpairable line in *file2*.

3 produce a line for each unpairable line in both *file1* and *file2*.

The normal output is also produced.

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

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

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

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

SEE ALSO

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

BUGS

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

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

NAME

kill – send a signal to a process, or terminate a process

SYNOPSIS

kill [–sig] processid ...
kill –l

DESCRIPTION

Kill sends the TERM (terminate, 15) signal to the specified processes. If a signal name or number preceded by '–' is given as first argument, that signal is sent instead of terminate (see *sigvec*(2)). The signal names are listed by using the –l option, and are as given in */usr/include/signal.h*, stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal, so **kill –9 ...** is a sure kill, as the KILL (9) signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (that is, processes resulting from the current login) are signaled (but beware: this works only if you use *sh*(1); not if you use *csh*(1).) The killed processes must belong to the current user unless he is the super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a TERM (terminate) signal by 'kill 1'; see *init*(8). To force *init* to close and open terminals according to what is currently in /etc/ttys use 'kill –HUP 1' (sending a hangup, signal 1).

The shell reports the process number of an asynchronous process started with '&' (run in the background). Process numbers can also be found by using *ps*(1).

Kill is built in to *csh*(1); it allows job specifiers, such as **kill %...**, in place of *kill* arguments. See *csh*(1) for details.

OPTIONS

–l Display a list of signal names.

SEE ALSO

csh(1), ps(1), kill(2), sigvec(2)

BUGS

An option to kill process groups ala *killpg*(2) should be provided; a replacement for **kill 0** for *csh*(1) users should be provided.

NAME

last – indicate last logins of users and teletypes

SYNOPSIS

last [*–number*][*–f filename*] [name ...] [tty ...]

DESCRIPTION

Last looks back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. *Last* displays the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

last reboot

will give an indication of mean time between reboot.

Last with no arguments displays a record of all logins and logouts, in reverse order.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) *last* indicates how far the search has progressed so far, and the search continues.

OPTIONS

–number

limit the number of entries displayed to that specified by *number*.

–f filename

Use *filename* as the name of the accounting file instead of */etc/wtmp*.

FILES

/usr/adm/wtmp	login data base
/usr/adm/shutdownlog	which records shutdowns and reasons for same

SEE ALSO

utmp(5), ac(8), lastcomm(1)

NAME

lastcomm – show last commands executed in reverse order

SYNOPSIS

lastcomm [command name] ... [user name] ... [terminal name] ...

DESCRIPTION

Lastcomm gives information on previously executed commands. *Lastcomm* with no arguments displays information about all the commands recorded during the current accounting file's lifetime. If called with arguments, *lastcomm* only displays accounting entries with a matching command name, user name, or terminal name.

EXAMPLES

tutorial% **lastcomm a.out root ttyd0**

would produce a listing of all the executions of commands named *a.out*, by user *root* while using the terminal *ttyd0*. and

tutorial% **lastcomm root**

would produce a listing of all the commands executed by user *root*.

For each process entry, *lastcomm* displays the following items of information:

- The command name under which the process was called.
- One or more flags indicating special information about the process. The flags have the following meanings:
  - F The process performed a *fork* but not an *exec*.
  - S The process ran as a set-user-id program.
  - D The process dumped memory.
  - X The process was killed by some signal.
- The name of the user who ran the process.
- The terminal which the user was logged in on at the time (if applicable).
- The amount of CPU time used by the process (in seconds).
- The date and time the process exited.

FILES

/usr/adm/acct accounting file

SEE ALSO

last(1), The name of the user who ran the process.
Flags, as accumulated by the accounting facilities in the system.
The command name under which the process was called.
The amount of cpu time used by the process (in seconds).
The time the process exited. sigvec(2), acct(5), core(5)

NAME

ld – link editor

SYNOPSIS

ld [**-A** *name*] [**-D** *hex*] [**-d**] [**-e** *entry*] [**-l**] [**-M**] [**-N**] [**-n**] [**-o** *name*] [**-r**] [**-S**] [**-s**] [**-T** *hex*] [**-t**] [**-u** *name*] [**-X**] [**-x**] [**-y***symbol*] [**-z**] file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. In the latter case, the **-r** option must be given to preserve the relocation bits. The output of *ld* is left on a file called *a.out* if not otherwise specified. The output file is made executable only if no errors occurred during the load.

The argument *files* are concatenated in the order specified. The entry point of the output is the beginning of the first routine unless the **-e** option is specified.

If any *file* is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. The first member of a library should be a file named '__.SYMDEF', which is understood to be a dictionary for the library as produced by *ranlib*(1); the dictionary is searched iteratively to satisfy as many references as possible.

The symbols **_etext**, **_edata** and **_end** (**etext**, **edata** and **end** in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

OPTIONS

Options should appear before the file names, except abbreviated library names specified by the **-l** option which can appear anywhere.

-A *name*
: Incremental loading: linking is to be done in a manner so that the resulting object may be read into an already executing program. The next argument is the name of a file whose symbol table is taken as a basis on which to define additional symbols. Only newly linked material is entered into the text and data portions of **a.out**, but the new symbol table will reflect every symbol defined before and after the incremental load. This argument must appear before any other object file in the argument list. The **-T** option may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding address (which must be a multiple of the pagesize). The default value is the old value of _end.

-D *hex* Pad the data segment with zero bytes until the data segment is *hex* bytes long.

-e *entry*
: Define the entry point: *entry* argument is the name of the entry point of the loaded program.

-l*x* This option is an abbreviation for the library name '/lib/lib*x*.a', where *x* is a string. If that does not exist, *ld* tries '/usr/lib/lib*x*.a' A library is searched when its name is encountered, so the placement of a **-l** is significant.

-M Produce a primitive load map, listing the names of the files which will be loaded.

-N Do not make the text portion read only or sharable. (Use 'magic number' 0407.)

-n Arrange (by giving the output file a 0410 'magic number') that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible segment boundary following

the end of the text.

-o *name*
Name is the name of the *ld* output file, instead of **a.out**.

-r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

-d Force definition of common storage even if the **-r** flag is present.

-S Strip the output by removing all symbols except locals and globals.

-s Strip the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

-T *hex* Start the text segment origin at location *hex*.

-t Trace: display the name of each file as it is processed.

-u *name*
Enter *name* as an undefined symbol. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-X Save local symbols except for those whose names begin with 'L'. This option is used by *cc*(1) to discard internally-generated labels while retaining symbols local to routines.

-x Do not preserve local (non-**.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

-y*sym* Display each file in which *symbol* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. It is usually necessary to begin *symbol* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.

-z Arrange for the process to be loaded on demand from the resulting executable file (0413 'magic number') rather than preloaded. This is the default. Results in a page-sized header on the output file followed by a text and data segment each of which have size a multiple of page-size bytes (being padded out with nulls in the file if necessary). With this format the first few BSS segment symbols may actually end up in the data segment; this is to avoid wasting the space resulting from data segment size roundup.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
/usr/local/lib/lib*.a	still more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), ranlib(1), strip(1)

BUGS

There is no way to force data to be page aligned.

NAME

leave – remind you when you have to leave

SYNOPSIS

leave [[+]hhmm]

DESCRIPTION

Leave sets an alarm to a time you specify and will tell you when the time is up. *Leave* waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. *Leave* disappears after you log off.

You can specify the time in on of two ways, namely as an absolute time of day in the form *hhmm* where *hh* is a time in hours (on a 12 or 24 hour clock), or you can place a **+** sign in front of the time, in which case the time is relative to the current time, that is, the specified number of hours and minutes from now. All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, *leave* prompts with "When do you have to leave?". *Leave* exits if you just type a newline, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a *.login* or *.profile*.

Leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use **kill -9** and its process id.

SEE ALSO

calendar(1)

EXAMPLES

The first example sets the alarm to an absolute time of day:

```
tutorial% leave 1535
Alarm set for Wed Mar  7 15:35:07 1984

                    work  work  work  work

tutorial% Time to leave!
```

The second example sets the alarm for 10 minutes in the future:

```
tutorial% leave +10
Alarm set for Wed Mar  7 15:45:24 1984

                    work  work  work  work

tutorial% Time to leave!

                    work  work  work  work

tutorial% You're going to be late!
```

NAME

lex – generator of lexical analysis programs

SYNOPSIS

lex [**–tvfn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analyis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, 'lex.yy.c' is generated, to be compiled thus:

```
cc lex.yy.c –ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

OPTIONS

–t Place the result on the standard output instead of in file *lex.yy.c*.

–v Print a one-line summary of statistics of the generated analyzer.

–n Opposite of –v; –n is default.

–f 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

EXAMPLE

```
lex lexcommands
```

would draw *lex* instructions from the file *lexcommands,* and place the output in *lex.yy.c*

```
%%
[A–Z] putchar(yytext[0]+ 'a'–'A');
[ ]+$
[ ]+   putchar(' ');
```

is an example of *lex*. This program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

SEE ALSO

yacc(1), sed(1)
The paper, *LEX – A Lexical Analyzer Generator,* in the Sun *Programming Tools* Manual.

NAME

lint – a C program verifier

SYNOPSIS

lint [**-abchnuvx**] [**-D** *name=def*] [**-D** *name*] [**-U** *name*] [**-I** *dir*] file ...
lint [**-C***lib*] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers.

Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as **-lm**, in the style of *ld*(1). The standard C library (**-lc**) is *lint*'ed by default. Arguments ending in *.ln* are also treated as library files.

To create lint libraries, use the **-C** option:

lint -Ccongress files . . .

where *files* are the C sources of library *congress*. The result is a file *llib-lcongress.ln* in the correct library format suitable for linting programs using **-lcongress**.

OPTIONS

h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

b Report *break* statements that cannot be reached. This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.

v Suppress complaints about unused arguments in functions.

x Report variables referred to by **extern** declarations, but never used.

a Report assignments of **long** values to **int** variables.

c Complain about casts which have questionable portability.

u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).

n Do not check compatibility against the standard library.

s Do not complain about structures that are never defined (e.g. using a structure pointer without knowing its contents.).

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

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

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

-C*lib* create a *lint* library with name *lib* (see DESCRIPTION section).

-l*lib* use *lint* library *lib* from the */usr/lib/lint* directory.

General Comments

The routine *exit*(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code.

/*VARARGS*n*/
: suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*NOSTRICT*/
: shuts off strict type checking in the next expression.

/*ARGSUSED*/
: turns on the **-v** option for the next function.

/*LINTLIBRARY*/
: at the beginning of a file shuts off complaints about unused functions in this file.

EXAMPLE

The following *lint* call:

```
lint -b myfile
```

checks the consistency of the file 'myfile'. The **-b** option indicates that unreachable **break** statements are to be checked.

FILES

/usr/lib/lint/lint[12]	programs
/usr/lib/lint/llib-lc.ln	declarations for standard functions
/usr/lib/lint/llib-lc	human-readable version of above
/usr/lib/lint/llib-lm.ln	declarations for math functions
/usr/lib/lint/llib-lm	human-readable version of above
/usr/lib/lint/llib-lmp.ln	declarations for multi-precision functions
/usr/lib/lint/llib-lmp	human-readable version of above
lib-l*.ln	library created with **-C**.

SEE ALSO

cc(1)

Lint, a C Program Checker, in *Programming Tools for the Sun Workstation*.

BUGS

There are some things you just *can't* get *lint* to shut up about.

NAME

ln – make links

SYNOPSIS

ln [**–f**] [**–s**] name1 [name2]
ln name ... directory

DESCRIPTION

A link is a directory entry referring to a file or another directory; the same file or directory (together with its size, all its protection information, etc.) may have several links to it. There are two kinds of links: hard links and symbolic links.

Ln makes hard links by default. A hard link to a file or directory is indistinguishable from the original directory entry; any changes to a file or directory are effective independent of the name used to reference the file or directory. Hard links may not span file systems.

Given one or two arguments, *ln* creates a link to an existing file or directory *name1*. If *name2* is given, the link has that name; *name2* may also be a directory in which to place the link; otherwise it is placed in the current directory. If *name2* is omitted or is a directory, the link made will have the same name as the last component of *name1*.

Given more than two arguments, the last argument must be the name of a directory. In this case, *ln* makes links to all the named files in the named directory. The links made will have the same name as the files being linked to.

OPTIONS

–f Force a hard link to a directory. The **–f** option is only available to the super-user.

–s Create symbolic links. A symbolic link contains the name of the file or directory to which it is linked. The referenced file or directory is used when an *open*(2) operation is performed on the link. A *stat*(2) on a symbolic link returns the linked-to file; an *lstat*(2) must be done to obtain information about the link. The *readlink*(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

EXAMPLES

The commands below illustrate the effects of the different forms of the *ln* command.

```
mars% ls –F                          See what files we've got
grab    jones/
mars% ls –F jones                    See what files there are in jones
house                                One file
mars% ln grab                        try to link a file in the same directory
./grab: File exists                  Sorry — can't link a file to itself
mars% ln jones/house                 link a file from another directory to here
mars% ls –F
grab    house   jones/
mars% ln grab hold                   link a file to another name in this directory
mars% ls –F
grab    hold    house   jones/
mars% ln grab hold jones             link files from here to jones
mars% ls –F jones
grab    hold    house
mars%
```

SEE ALSO

rm(1), cp(1), mv(1), link(2), readlink(2), stat(2), symlink(2)

BUGS

Error messages print the wrong file name when the **–s** option is used.

NAME

lockscreen – maintain window context until "login"

SYNOPSIS

lockscreen [–e] [–r]

DESCRIPTION

Lockscreen maintains the user's window setup and context. The user does not need to log out from the machine. When run with the machine not in use, the black *lockscreen* display and constantly moving Sun Microsystems logo limit phosphorus burn of the video display that may occur from running the same window configuration for a long time.

Lockscreen should be executed from a terminal emulator running inside the SunWindows system. The window described by the environment parameter WINDOW_PARENT (such as, */dev/win0*) is covered and obscured by black. This window is typically the entire screen. When any keyboard or mouse button is pressed, the black screen is replaced by an option and message subwindow. The message subwindow (the top subwindow) prompts the user to fill in the *Name* and *Password* text items in the option subwindow (the bottom subwindow). See *suntools*(1) for a general description of interacting with option subwindows. Only *root* and the user name that invoked *lockscreen* are acceptable names. Initially, the user name is filled in and the password is the text item to which typed characters are directed.

Once the name and password are entered, the user can use the mouse to choose the *Show Desktop* command item. If the name and password are valid, *lockscreen* terminates and reveals the windows under it.

At any time, the user can choose the *Abort Login* command item, which replaces the subwindows with the dark background and the logo.

Also, at any time, any user can choose the *Exit Desktop* command item, which is analogous to choosing the *Exit* menu item in the Root Manager menu of the *suntools*(1) program. The program from which *suntools*(1) executed is then in control of the machine. Notice that if *suntools*(1) was started from some program other than *login*(1), whoever is sitting at the terminal is now logged in as the person that started *suntools*(1). To prevent a breach of security, the –e command line switch can be given to *lockscreen* to suppress the display of the *Exit Desktop* command item.

OPTIONS

–e Add the display of the *Exit Desktop* command item. The default is no exit option.

–r Someone correctly logging in as *root* may invoke the *Show Desktop* command option. The default is that this feature is disabled.

SEE ALSO

suntools(1), login(1)

NAME

login – sign on

SYNOPSIS

login [username]

DESCRIPTION

Login signs *username* on to the system initially; *login* may also be used at any time to change from one user i.d. to another.

If you use *login* without an argument, *login* requests a user name, and a password if appropriate. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated, the user is informed of the existence of mail, and the message of the day is printed, as is the time he last logged in (unless he has a *.hushlogin* file in his home directory – this is mainly used to make life easier for non-human users, such as *uucp*).

Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(1)) according to specifications found in a password file. Argument 0 of the command interpreter is "–sh", or more generally the name of the command interpreter with a leading dash ("–") prepended.

Login also initializes the environment *environ*(5) with information specifying home directory, command interpreter, terminal type (if available) and user name.

If the file */etc/nologin* exists, *login* prints its contents on the user's terminal and exits. This is used by *shutdown*(8) to stop logins when the system is about to go down.

Login is recognized by *sh*(1) and *csh*(1) and executed directly (without forking).

Login times out and exits if its prompt for input is not answered within some 'reasonable' time.

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/spool/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/nologin	stops logins
.hushlogin	makes login quieter
/etc/securetty	lists ttys that root may log in on

SEE ALSO

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

DIAGNOSTICS

"Login incorrect," if the name or the password is bad (or mis-typed).
"No Shell", "cannot open password file", "no directory": consult a programming counselor.

NAME

look – find lines in a sorted list

SYNOPSIS

look [–df] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*.

OPTIONS

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

–f Fold: Upper case letters compare equal to lower case.

If no *file* is specified, *look* uses */usr/dict/words* with collating sequence **–df**.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

Give *lorder* one or more object or library archive (see *ar*(1)) *files*, and it lists pairs of object file names — meaning that the first file of the pair refers to external identifiers defined in the second — to the standard output. *Lorder*'s output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

EXAMPLE

This brash one-liner intends to build a new library from existing *.o* files.

```
ar cr library `lorder *.o | tsort`
```

The *ranlib*(1), command converts an ordered archive into a randomly accessed library and makes *lorder* unnecessary.

SEE ALSO

tsort(1), ld(1), ar(1), ranlib(1)

BUGS

The names of object files, in and out of libraries, must end with '.o'; otherwise, nonsense results.

NAME

lpq – spool queue examination program

SYNOPSIS

lpq [+[*num*]] [**–l**] [**–P***printer*] [*job #* ...] [*user* ...]

DESCRIPTION

lpq examines the spooling area used by *lpd*(8) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user.

Lpq reports on any jobs currently in the queue when invoked without any options. See the OPTIONS section below for a list of options. Arguments supplied that are not recognized as options are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (that is, invocation of *lpr*(1)) *lpq* reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to *lprm*(1) for removing a specific job), and the total size in bytes. The **–l** option causes information about each of the files comprising the job to be printed. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when *lpr*(1) is used as a sink in a pipeline) in which case the file is indicated as '(standard input)'.

If *lpq* warns that there is no daemon present (that is, due to some malfunction), the *lpc*(8) command can be used to restart the printer daemon.

OPTIONS

Lpq reports on any jobs currently in the queue when invoked without any options.

–P*printer*
: route the output to the printer specified by *printer*. In the absence of the **–P** option, the default line printer is used (or the value of the PRINTER variable in the environment).

+*nnn* display the spool queue until it empties. Supplying a number *nnn* immediately after the **+** sign indicates that *lpq* should sleep *nnn* seconds in between scans of the queue.

FILES

/etc/termcap	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/*	the spooling directory, as determined from printcap
/usr/spool/*/cf*	control files specifying jobs
/usr/spool/*/lock	the lock file to obtain the currently active job

SEE ALSO

lpr(1), lprm(1), lpc(8), lpd(8)

BUGS

The **+** option doesn't wait until the entire queue is empty; it only waits until the local machine's queue is empty.

Due to the dynamic nature of the information in the spooling directory *lpq* may report unreliably.

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

lpq is sometimes unable to open various files because the lock file is malformed.

DIAGNOSTICS

waiting for *printer* **to become ready**

The daemon could not open the printer device. This can happen for a number of reasons; the most common is that the printer is turned off-line. This message can also be generated if the printer is out of paper, the paper is jammed, and so on. The actual reason is dependent on the meaning of error codes returned by system device driver. Not all

printers supply sufficient information to distinguish when a printer is off-line or having trouble (for example, a printer connected through a serial line). Another possible cause of this message is some other process, such as an output filter, has an exclusive open on the device. Your only recourse here is to kill off the offending program(s) and restart the printer with *lpc*.

***printer* is ready and printing**

The *lpq* program checks to see if a daemon process exists for *printer* and prints the file *status*. If the daemon is hung, a super user can use *lpc* to abort the current daemon and start a new one.

waiting for *host* to come up

Indicates that there is a daemon trying to connect to the remote machine named *host* in order to send the files in the local queue. If the remote machine is up, *lpd* on the remote machine is probably dead or hung and should be restarted as mentioned for *lpr*.

sending to *host*

The files should be in the process of being transferred to the remote *host*. If not, the local daemon should be aborted and started with *lpc*.

Warning: *printer* is down

The printer has been marked as being unavailable with *lpc*.

Warning: no daemon present

The *lpd* process overseeing the spooling queue, as indicated in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. The error log file for the printer should be checked for a diagnostic from the deceased process. To restart an *lpd*, use

% **lpc restart** *printer*

NAME

lpr – off line print

SYNOPSIS

lpr [**-P***printer*] [**-#***num*] [**-C***class*] [**-J***job*] [**-T***title*] [**-i**[*num*]] [**-1234***font*] [**-w***num*] [**-plotndgvcfrmhs**] [filename ...]

DESCRIPTION

Lpr uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed.

OPTIONS

-P Force output to the named *printer*. Normally, the default printer is used (site dependent), or the value of the environment variable PRINTER is used.

-# Produce multiple copies of output, using *num* as the number of copies for each file named. For example,

```
% lpr -#3 foo.c bar.c more.c
```

produces three copies of the file *foo.c*, followed by three copies of *bar.c*, etc. On the other hand,

```
% cat foo.c bar.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,

```
% lpr -C EECS foo.c
```

replaces the system name (the name returned by *hostname*(1)) with EECS on the burst page, and prints the file *foo.c*.

-J Print *job* as the job name on the burst page. Normally, *lpr* uses the first file's name.

-T Use *title* instead of the file name for the title used by *pr*(1).

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

-1234 Mount the specified *font* on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size*.

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

The following single letter options are used to notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

-p Use *pr*(1) to format the files (equivalent to *print*).

-l Print control characters and suppress page breaks.

-t The files contain data from *troff*(1) (cat phototypesetter commands).

-n The files contain data from *ditroff* (device independent troff).

-d The files contain data from *tex*(l) (DVI format from Stanford).

-g The files contain standard plot data as produced by the *plot*(3X) routines (see also *plot*(1G) for the filters used by the printer spooler).

-v The files contain a raster image for devices like the Benson Varian.

-c The files contain data produced by *cifplot*(l).

-f Interpret the first character of each line as a standard FORTRAN carriage control character.

The remaining single letter options are:

-r Remove the file upon completion of spooling.

-m Send mail upon completion.

-h Suppress the printing of the burst page.

-s Use *symlink*(2) to link data files rather than trying to copy them (so large files can be printed). This means the files should not be modified or removed until they have been printed.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities data base
/usr/lib/lpd*	line printer daemons
/usr/spool/*	directories used for spooling
/usr/spool/*/cf*	daemon control files
/usr/spool/*/df*	data files specified in "cf" files
/usr/spool/*/tf*	temporary copies of "cf" files

SEE ALSO

lpq(1), lprm(1), pr(1), symlink(2), printcap(5), lpc(8), lpd(8)

DIAGNOSTICS

lpr: *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). Use

% ps ax | fgrep lpd

to get a list of process identifiers of running lpd's. The *lpd* to kill is the one which is not listed in any of the "lock" files (the lock file is contained in the spool directory of each printer). Kill the master daemon using the following command.

% kill *pid*

Then remove */dev/printer* and restart the daemon (and printer) with the following commands.

% rm /dev/printer % /usr/lib/lpd

Another possibility is that the *lpr* program is not setuid *root*, setgid *spooling*. This can be checked with

% ls -lg /usr/ucb/lpr

lpr: *printer*: printer queue is disabled

This means the queue was turned off with

% 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*.

NAME

lprm – remove jobs from the line printer spooling queue

SYNOPSIS

lprm [**–P***printer*] [–] [*job #* ...] [*user* ...]

DESCRIPTION

Lprm removes a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

Lprm without any arguments will delete the currently active job if it is owned by the user who invoked *lprm*.

If the – flag is specified, *lprm* will remove all jobs which a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the *lpr* command was invoked.

Specifying a user's name, or list of user names, will cause *lprm* to attempt to remove any jobs queued belonging to that user (or users). This form of invoking *lprm* is useful only to the super-user.

A user may dequeue an individual job by specifying its job number. This number may be obtained from the *lpq*(1) program. For example:

```
tutorial% lpq –Pimagen
imagen is ready and printing
Rank   Owner   Job  Files                 Total Size
active wendy   385  standard input        35501 bytes
tutorial% lprm –Pimagen 305
```

Lprm announces the names of any files it removes and is silent if there are no jobs in the queue which match the request list.

Lprm will kill off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The **–P** option may be used to specify the queue associated with a specific printer (otherwise the default printer, or the value of the PRINTER variable in the environment is used).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently active job

SEE ALSO

lpr(1), lpq(1), lpd(8)

DIAGNOSTICS

lprm: *printer*: **cannot restart printer daemon**

The connection to *lpd* on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket */dev/printer* to be sure it still exists (if it does not exist, there is no *lpd* process running). Use

% ps ax | fgrep lpd

to get a list of process identifiers of running lpd's. The *lpd* to kill is the one which is not listed in any of the "lock" files (the lock file is contained in the spool directory of each printer). Kill the master daemon using the following command.

% kill *pid*

Then remove */dev/printer* and restart the daemon (and printer) with the following commands.

% rm /dev/printer % /usr/lib/lpd

Another possibility is that the *lpr* program is not setuid *root*, setgid *spooling*. This can be checked with

% ls -lg /usr/ucb/lpr

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

ls – list contents of directory

SYNOPSIS

ls [**–acdfgilqrstu1ACLFR**] name ...

DESCRIPTION

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

OPTIONS

There are a large number of options:

–l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by "->".

–g Include the group ownership of the file in a long output.

–t Sort by time modified (latest first) instead of by name.

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

–s Give size in kilobytes of each file.

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

–L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

–r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

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

–c Use time of file creation for sorting or printing.

–i For each file, print the i-number in the first column of the report.

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

–F Mark directories with a trailing '/', symbolic links with a trailing '@', and executable files with a trailing '*'.

–R Recursively list subdirectories encountered.

–1 Force one entry per line output format; this is the default when output is not to a terminal.

–C Force multi-column output; this is the default when output is to a terminal.

–q Force printing of non-graphic characters in file names as the character '?'; this is the default when output is to a terminal.

INTERPRETATION OF LISTING

The mode printed under the –l option contains 11 characters which are interpreted as follows: the first character is

d if the entry is a directory;

b if the entry is a block-type special file;
c if the entry is a character-type special file;
l if the entry is a symbolic link
s if the entry is an AF_UNIX domain socket, or
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
- if the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the user-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally 'x' or '-') is **t** if the 1000 bit of the mode is on. See *chmod*(1) for the meaning of this mode.

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

FILES

/etc/passwd to get user id's for "*ls* **-l**".
/etc/group to get group id's for "*ls* **-g**".

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as "*ls* **-s**" is much different than "*ls* **-s** | lpr". On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

NAME

m4 – macro processor

SYNOPSIS

m4 [file ...]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each argument file is processed in order; the standard input is read if there are no arguments or if an argument is '–'. The processed text is written on the standard output.

Macro calls have the form

name(arg1,arg2, . . . , argn)

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

Left and right single quotes (` ') are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $n in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string.

undefine removes the definition of the macro named in its argument.

ifdef If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX versions of *m4*.

changequote
Change quote characters to the first and second arguments. *Changequote* without arguments restores the original values (that is, ` ').

divert *M4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum returns the value of the current output stream.

dnl reads and discards characters up to and including the next newline.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation); relationals; parentheses.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

substr returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is identical to *include,* except that it says nothing if the file is inaccessible.

syscmd executes the UNIX command given in the first argument. No value is returned.

maketemp fills in a string of XXXXX in its argument with the current process id.

errprint prints its argument on the diagnostic output file.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.

SEE ALSO

M4 — A Macro Processor , in *Programming Tools for the Sun System.*

NAME

mail – send and receive mail

SYNOPSIS

mail [**–v**] [**–f** [*name*]] [*people* ...]

INTRODUCTION

Mail is an intelligent, interactive facility for handling electronic mail. *Mail*'s basic unit of information is a *message*. *Mail* provides facilities for sending mail to other people, replying to messages you receive from other people, and for browsing through received messages.

Sending mail. To send a message to one or more other *people*, use *mail* with arguments which are the names of *people* to send to. The *people* arguments can be the names of users on your own system, or they can be network addresses — see below. Then, type in your message, followed by an EOT (control–D) at the beginning of a line. The section labelled *Replying to or originating mail,* describes some features of *mail* available to help you compose your letter.

The **–v** option is a debugging aid for *mail* itself: it lets you see if your mail is being sent out OK.

Reading mail. If you use *mail* without arguments, it checks your mail out of the post office, and then displays a one-line header of each message found there. The 'current message' is initially the first message (numbered 1); you can display it with the **print** command (abbreviated **p**). You can move among the messages: use the '+' command to move forward a message, the '–' command to move back a message, and simple numbers to address a particular message.

Disposing of mail. After examining a message you can **delete** (**d**) the message or **reply** (**r**) to it. Deleting a message makes *mail* forget about it. If you accidentally delete a message, you do have recourse: you can **undelete** (**u**) the message by giving its number, or you can **exit** (**x**) the *mail* session, leaving everything as it was before you started the session. Messages deleted during the *mail* session however, usually disappear and are never seen again.

Specifying messages. Commands such as **print** and **delete** can be given a list of message-numbers as arguments to apply to a number of messages at once. Numbers in a list are separated by spaces. Message-numbers are simple numbers, or they can be specified as a range, that is, two numbers separated by a minus sign. Thus:

delete 1 2

deletes messages 1 and 2, while:

delete 1–5

deletes messages 1 through 5. The special name '*' addresses *all* messages, and '$' addresses the *last* message; thus the **top** command (print the first few lines of a message) could be used in:

top *

to print the first few lines of all messages.

Replying to or originating mail. You can use the **reply** (**r**) command to respond to mail from someone. **r** works somewhat like

% **mail person**

but **reply** already knows that **person** is the person who sent you the message you're replying to: type in the text of your message, and end the text with an EOT at the beginning of a line. There are some variations of the **reply** command:

r (the straightforward **reply** command) replies to the person who sent you the message. This command can also be typed as **replysender**.

R (upper-case **R**) replies to everyone who got the original message. This command can also be typed as **replyall**.

Also see the **replyall** variable.

While you are composing a message, you can use 'escapes' (usually called 'tilde escapes') to get *mail* to treat such lines in special ways. For instance, typing '~m' (alone on a line) places a copy of the current message into the response, but shifted right by one tabstop. Other escapes set up subject fields, add and delete recipients to the message, and allow you to escape to an editor to

revise the message or to a shell to run some commands. These options are described in the summary below.

Ending a mail processing session. You end a *mail* session with the **quit (q)** command. Messages which have been examined but not deleted go to your *mbox* file. Messages which have been deleted are discarded. Unexamined messages go back to the post office. The **-f** option specifies that *mail* read in the contents of your *mbox* (or the specified file) for processing; when you **quit** *mail* writes undeleted messages back to this file.

Personal and systemwide distribution lists. You can create personal distribution lists so that, for instance, you can send mail to 'cohorts' and have it go to a group of people. Such lists can be defined by placing a line like:

alias cohorts bill ozalp sklower jkf mark cory:kridle

in the file *.mailrc* in your home directory. The current list of such aliases can be displayed by the **alias (a)** command in *mail*.

System Distribution Lists System wide distribution lists can be created by editing */usr/lib/aliases*, see *aliases*(5) and *sendmail*(8); these are kept in a slightly different syntax. In mail you send, personal aliases are expanded in mail sent to others so that they can **reply** (lower case **r**) to the recipients. System-wide aliases are not expanded when the mail is sent, but any reply returned to the machine will have the system wide alias expanded as all mail goes through *sendmail*. If you edit */usr/lib/aliases*, you must run the program *newaliases*(8) to rebuild the aliases database.

Network mail (ARPA, UUCP) Mail to sites on the ARPA or UUCP network can be sent using *name@site* for ARPA-net sites or *machine!user* for UUCP sites, provided appropriate gateways are known to the system. Be sure to escape the ! in UUCP sites when giving it on a *csh* command line by preceding it with a \.

Mail has a number of options which can be **set** in the *.mailrc* file to alter its behavior; thus

set askcc

enables the 'askcc' (ask for carbon copy) feature. These options are summarized below in the section labelled *Mail Options*.

SUMMARY OF MAIL COMMANDS

This summary is adapted from the *Mail User's Guide*.

Each *mail* command is typed on a line by itself. Arguments may follow the command word. The command word need not be typed in its entirety — the first command which matches the typed prefix is used. Commands which take message lists as arguments use the next message forward which satisfies the command's requirements if a message-list is not specified. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, *mail* types 'No applicable messages' and aborts the command.

-[*nnn*] Go to the previous message and print it out. If the optional numeric argument *nnn* is specified, go to the *nnn*'th previous message and print it.

+[*nnn*]
Go to the next message and print it out. If the optional numeric argument *nnn* is specified, go to the *nnn*'th next message and print it.

? Print a brief summary of commands.

! *Shell Command*
Execute the UNIX *Shell Command* which follows the **!**.

alias **(a)** With no arguments, print out all currently-defined aliases. With one argument, print out that alias. With more than one argument, add the users named in the second and later arguments to the alias named in the first argument.

chdir [*directory*]

(c) Change the user's working directory to the optional *directory* argument. Change to the user's login directory if *directory* is not specified.

delete (d) Takes a list of messages as argument and marks them all as deleted. Deleted messages are not saved in *mbox*, nor are they available for most other commands. Messages deleted in the current *mail* session can be undeleted with the **undelete** command described later.

dp (also **dt**) Delete the current message and print the next message. If there is no next message, *mail* says 'at EOF.'

edit (e) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.

exit (**ex** or **x**) Effects an immediate return to the Shell without modifying the user's system mailbox, the *mbox* file, or the edit file when using the **–f** option.

from (f) Takes a list of messages and prints their message headers.

headers
(h) List the current range of headers, 20 at a time (fewer lines are displayed on low-speed terminals). If you want to see another group of headers, you must type a number that lies within that group. For example,

headers

alone displays the headers of the first 20 messages. To see headers (say) 41 thru 60, you can type

headers 45

or in fact any number that lies in the range 41 thru 60. Also see the **z** command below, which scrolls forwards and backwards through groups of headers.

help A synonym for **?**

hold (**ho**, also **preserve**) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in *mbox*. Does not override the **delete** command.

ignore takes a list of character strings as arguments. Lines in the header of mail messages that start with any of the specified character strings are ignored.

mail (m) Takes as arguments login names and distribution group names and sends mail to those people.

next (**n** like **+** or CR) Goes to the next message in sequence and types it. With an argument list, types the next matching message.

preserve
A synonym for **hold.**

print (p) Takes a message list and types out each message on the user's terminal. Typing a **print** command with a capital **P** prints even ignored headers.

quit (q) Terminate the *mail* session, saving all undeleted, unsaved messages in your *mbox* file in your login directory, preserving all messages marked with **hold** or **preserve** or never referenced in your system mailbox, and removing all other messages from your system mailbox. If new mail has arrived during the *mail* session, *mail* displays a message:

You have new mail

If you **quit** while editing a mailbox file with the **–f** flag, *mail* rewrites the edit file. *Mail* then returns to the Shell, unless the rewrite of edit file fails, in which case the user can escape with the **exit** command.

reply (r) Takes a message list and sends mail to each message author just like the **mail** command. The default message must not be deleted.

replyall

Replies to all members on a distribution list, regardless of the setting of the **replyall** variable.

replysender
Reply to the sender of the message, regardless of the setting of the of the **replyall** variable.

Reply (upper-case **R**) replies to all members on a distribution list.

respond
A synonym for **reply**.

Respond
A synonym for **Reply**.

save (**s**) Takes a message list and a filename and appends each message in turn to the end of the file. The filename in quotes, followed by the line count and character count is echoed on the user's terminal.

set (**se**) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form:

option=value

or:

option

shell (**sh**) Invokes an interactive version of the shell.

size Takes a message list and prints out the size in characters of each message.

subject
(**su**) works like the **from** command but works with subjects instead of message headers.

top Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to five.

type (**t**) A synonym for **print**.

unalias
Takes a list of names defined by **alias** commands and discards the remembered groups of users. The group names no longer have any significance.

undelete
(**u**) Takes a message list and marks each one as *not* being deleted.

unset Takes a list of option names and discards their remembered values; the inverse of **set**.

visual (**v**) Takes a message list and invokes the display editor on each message.

write (**w**) is similar to the **save** command, but **write** removes the 'From' line and the 'Status' line from the header of the messages, whereas **save** leaves those lines in the message.

xit (**x**) A synonym for **exit**.

z Scrolls forwards and backwards through groups of headers, 20 at a time (fewer lines of headers on low-speed terminals). If a '+' argument is given, the next 20 headers are displayed, and if a '–' argument is given, the previous 20 headers are displayed.

ESCAPE SEQUENCES

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name 'tilde escape' is somewhat of a misnomer since the actual escape character can be set by the option **escape**.

~!command
Execute the indicated shell command, then return to the message.

~c name ...

Add the given names to the list of carbon copy recipients.

~d Read the file *dead.letter* from your home directory into the message.

~e Invoke the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~f messages
Read the named messages into the message being sent. If no messages are specified, read the current message. Also see the **~m** command which does the same thing but shifts the message right by one tab stop.

~h Edit the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field by using the current terminal erase and kill characters.

~m messages
Read the named messages into the message being sent, shifted right one tab. If no messages are specified, read the current message.

~p Print out the message collected so far, prefaced by the message header fields.

~q Abort the message being sent, copying the message to *dead.letter* in your home directory if **save** is set.

~r filename
Read the named file into the message.

~s string
Make the named string become the current subject field.

~t name ...
Add the given names to the direct recipient list.

~v Invoke an alternate editor (defined by the VISUAL option) on the message collected so far. Usually, the alternate editor is a screen editor. After you quit the editor, you may resume appending text to the end of your message.

~w filename
Write the message onto the named file.

~|command
Pipe the message through the command as a filter. If the command gives no output or terminates abnormally, retain the original text of the message. The command *fmt*(1) is often used as *command* to rejustify the message.

~~string
Insert the string of text in the message prefaced by a single ~. If you have changed the escape character, then you should double that character in order to send it.

MAIL OPTIONS

Options are controlled via the **set** and **unset** commands. Options may be either binary, in which case it is only significant to see whether they are set or not, or string, in which case the actual value is of interest. You can set and unset options during a *mail* session, or you can place the options in your *.mailrc* file in your home directory.

Binary Options. The binary options include the following:

append
Messages saved in *mbox* are appended (the default case) to the end rather than prepended. This is set in */usr/lib/Mail.rc* on version 7 systems.

ask *Mail* prompts you for the subject of each message you send. If you respond with simply a newline, no subject field is sent.

askcc *Mail* prompts for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

autoprint
The **delete** command behaves like **dp** — thus, after deleting a message, the next one is typed automatically.

ignore *Mail* ignores interrupt signals from your terminal and echoes them as @'s.

metoo Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting **metoo** includes the sender in the group.

nosave
Prevents *mail* from saving messages in the *dead.letter* file in your home directory on receipt of two interrupts (or after a ~**q**). The default action is to save such messages in the *dead.letter* file.

quiet Suppress printing the version of the *mail* program when first invoked.

replyall
Alters the action of the **reply** and **Reply** commands: **reply** (lower-case **r**) normally replies to just the sender of the message. **Reply** (upper-case **R**) normally replies to everyone on the distribution list. Setting the **replyall** variable reverses this behavior so that **r** (lower-case **r**) replies to everyone on the distribution list, and **R** (upper-case **R**) replies to just the sender. This feature is here to retain compatibility with older versions of the *mail* system.

String Options. The following options have string values:

EDITOR
Pathname of the text editor to use in the **edit** command and ~e escape. If not defined, then a default editor is used. The default editor is */usr/ucb/ex*.

SHELL Pathname of the shell to use in the **!** command and the ~! escape. A default shell is used if this option is not defined. The default shell is */bin/csh*.

VISUAL
Pathname of the text editor to use in the **visual** command and ~v escape.

escape If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.

record If defined, gives the pathname of the file used to record all outgoing mail. If not defined, outgoing mail is not saved.

toplines
If defined, gives the number of lines of a message to be printed out with the **top** command; normally, the first five lines are printed.

crt If defined, gives the number of lines of a message to be printed before *mail* calls up the *more*(1) utility so that you can peruse the mail messages leisurely.

EXAMPLES

Send mail to Wendy on her system called ariel:

```
angel% mail wendy@ariel
Subject: New DBX Manuals
The new DBX manual pages are in angel:/usr/man/man1/dbx.1
^D
EOT
angel%
```

Note that *mail* asked for a subject; this is controlled by the **ask** option in the *.mailrc* file. There is an example of such a file later on in this section.

Now when Wendy is working on her system, she will see a mail notification:

```
work, work, work
You have new mail.
ariel% mail
Mail version 2.17  12/26/82.  Type ? for help
"/usr/spool/mail/wendy":  1 message 1 new
>N  1  cuthbert  Thu Nov  3 10:03:01  11/265  "New DBX Manual Pages"
&                                   Mail signals its readiness for commands
                                    Carriage-return means read next message

From cuthbert Thu Nov  3 10:03:01 1983
Date: 3 Nov 83 10:02:56 PST (Thu)
From: cuthbert (Cuthbert Pouncetrifle)
Subject: New DBX Manual Pages
To: wendy
Status: R

The new DBX manual pages are in angel:/usr/man/man1/dbx.1

&                                   Mail signals its readiness for commands
                                    Carriage-return means read next message
At EOF                              Means there are no more messages
& d                                 Delete this message
& q                                 Quit from the mail program
ariel%
```

Here is a *.mailrc* file in which you can set options to control *mail*'s behavior:

```
set crt=30 ask
ignore via apparently-to date from status received message-id
```

The first line sets the **crt** option to 30 lines — if there are messages longer than this, *mail* calls up *more* to page through the message in 30-line chunks. The **ask** option makes *mail* ask for a subject heading every time you send mail to someone. Finally, the second line says to ignore any line starting with any of the character strings from the list — this shortens the amount of irrelevant junk that appears at the top of mail messages.

Here is a line from a *.login* file (oriented to the C-Shell):

```
set mail=(10 /usr/spool/mail/$USER)
```

This **set** command instructs the C-Shell to look in */usr/spool/mail/$USER* for mail every 10 seconds and to notify you if there is mail waiting there. In the absence of the time specification, the Shell looks every five minutes.

FILES

/usr/spool/mail/*	post office
~/mbox	your old mail
~/.mailrc	file giving initial mail commands
/tmp/R#	temporary for editor escape
/usr/lib/Mail.help*	help files
/usr/lib/Mail.rc	system initialization file
/bin/mail	to do actual mailing
/usr/lib/sendmail	postman

SEE ALSO

binmail(1), fmt(1), newaliases(8), aliases(5), sendmail(8), mailaddr(7), biff(1)
Mail User's Guide in the *Beginner's Guide to the Sun Workstation*, which gives an excellent introduction.

For *sendmail* installation instructions, see the Sun *System Manager's Manual*.

NAME

make – maintain program groups

SYNOPSIS

make [**–f** *makefile*] [**–i**] [**–k**] [**–n**] [**–t**] [**–r**] [**–s**] [**–d**] [**–p**] [**–S**] [**–q**] file ...

DESCRIPTION

Make executes commands in *makefile* to update one or more target *files*. *File* is typically a program. If no **–f** option is present, 'makefile' and 'Makefile' are tried in order. 'Makefile' with a capital M is the preferred choice since it appears towards the front of any list of files and thus tends to stand out. If *makefile* is '–', *make* reads the standard input. More than one **–f** option may appear.

Make updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target. If a name appears on the left of more than one 'colon' line, then it depends on all of the names on the right of the colon on those lines, but only one command sequence may be specified for it. If a name appears on a line with a double colon :: then the command sequence following that line is performed only if the name is out of date with respect to the names to the right of the double colon, and is not affected by other double colon lines on which that name may appear.

Two special forms of a name are recognized. A name like *library*(*file*) means the file named *file* stored in the archive named *library*. A name like *library*((*entry*)) means the file stored in archive *library* containing the entry point *entry*.

Sharp and newline surround comments.

The following makefile says that *program* depends on the two files *a.o* and *b.o* and that they in turn depend on *.c* files and a common file called *incl*:

```
program: a.o b.o
        cc a.o b.o –lm –o program
a.o: incl a.c
        cc –c a.c
b.o: incl b.c
        cc –c b.c
```

Makefile entries of the form

```
string1 = string2
```

are macro definitions. Subsequent appearances of $(*string1*) or ${*string1*} are replaced by *string2*. If *string1* is a single character, the parentheses or braces are optional.

Make infers prerequisites for files for which *makefile* gives no construction commands. For example, a *.c* file may be inferred as prerequisite for a *.o* file and be compiled to produce the *.o* file. Thus the preceding example can be done more briefly:

```
program: a.o b.o
        cc a.o b.o –lm –o program
a.o b.o: incl
```

Prerequisites are inferred according to selected suffixes listed as the 'prerequisites' for the special name .SUFFIXES; multiple lists accumulate; an empty list clears what came before. Order is significant; the first possible name for which both a file and a rule as described in the next paragraph exist is inferred. The default list is

```
.SUFFIXES: .out .o .c .e .r .f .y .l .s .p
```

The rule to create a file with suffix *s2* that depends on a similarly named file with suffix *s1* is specified as an entry for the 'target' *s1s2*. Before issuing any command, *make* sets certain special macros:

$@ is set to the name of the file to be 'made'.

$? is set to the names of files that are younger than the target.

$< is set to the name of the related file that caused the action, only if the command was generated by an implicit rule.

$* is set to the prefix shared by the current and dependent filenames, only if the command was generated by an implicit rule.

For example, a rule for making optimized *.o* files from *.c* files is

```
.c.o: ; cc -c -O -o $@ $*.c
```

Certain macros are used by the default inference rules to communicate optional arguments to any resulting compilations. In particular, 'CFLAGS' is used for *cc*(1) options, 'FFLAGS' for *f77*(1) options, 'PFLAGS' for *pc*(1) options, and 'LFLAGS' and 'YFLAGS' for *lex* and *yacc*(1) options. In addition, the macro 'MFLAGS' is filled in with the initial command line options supplied to *make*. This simplifies maintaining a hierarchy of makefiles as one may then invoke *make* on makefiles in subdirectories and pass along useful options such as **-k**.

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target .SILENT is in *makefile*, or the first character of the command is '@'.

Commands returning nonzero status (see *intro*(1)) cause *make* to terminate unless the special target .IGNORE is in *makefile* or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target is a directory or depends on the special name .PRECIOUS.

OPTIONS

-f *myfile*
: *myfile* is the name of the file to use for *make* commands instead of the default 'makefile' or 'Makefile'.

-i Equivalent to the special entry .IGNORE:.

-k When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

-n Trace and print, but do not execute the commands needed to update the targets.

-t Touch, that is, update the modified date of targets, without executing any commands.

-r Equivalent to an initial special entry .SUFFIXES: with no list.

-s Equivalent to the special entry .SILENT:.

-d debug; print internal state after each command

-p print internal state after reading Makefile

-q determines if the target is up to date — returns a zero exit status if it is, and a non-zero exit status otherwise.

-S undoes the effect of the **-k** option.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1), f77(1), pc(1)
Make - A Program for Maintaining Computer Programs, in *Programming Tools for the Sun*

System.

BUGS

Some commands return nonzero status inappropriately. Use –i to overcome the difficulty. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across newlines in *make*.

***Make* should recognize names of libraries in the dependency entries, for example:**

```
grab:   -lm grab.o
        cc -o grab grab.o -lm
```

NAME

man – print out manual pages; find manual information by keywords

SYNOPSIS

man [–] [–**t**] [section] title ...
man –**k** keyword ...
man –**f** file ...

DESCRIPTION

Man displays information from the UNIX System Manuals. *Man* can be asked for one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. It can also provide on-line access to the sections of the printed manual.

When neither the –**k** nor –**f** option is specified (see OPTIONS below), *man* formats a specified set of manual pages. If a *section* is specified, *man* looks in that section of the manual for the given *titles*. *Section* is an arabic section number ('3' for instance); the number may be followed by a single letter classifier ('1g', for instance, indicating a graphics program in Section 1). If *section* is omitted, *man* searches all sections of the manual, giving preference to commands over subroutines in system libraries, and prints the first section it finds, if any.

If the standard output is a terminal, or if you use the – flag, *man* pipes its output through *cat*(1) with the –**s** option to crush out useless blank lines, *ul*(1) to create proper underlines for different terminals, and through *more*(1) to stop after each page on the screen. Type a space to continue, and a control-D to scroll 11 more lines when the output stops.

OPTIONS

–**k** Display a one line synopsis of each manual section whose listing in the table of contents contains any of the *keywords*.

–**f** Attempt to locate manual sections related to those files, and display the table of contents lines for those sections.

–**t** Use *troff* to format the specified section, assuming you have a suitable raster output device which can actually handle *troff*'s output.

FILES

/usr/man/man?/*	manual pages in *nroff*/*troff* source
/usr/man/cat?/*	formatted manual pages

SEE ALSO

more(1), ul(1), whereis(1), catman(8)

BUGS

The manual is supposed to be reproducible either on the phototypesetter or on a typewriter. However, on a typewriter some information is necessarily lost.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1), talk(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir – make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates directories. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

The current *umask*(2) setting determines the mode in which directories are created. Modes may be modified after creation by using *chmod*(1).

Mkdir requires write permission in the parent directory.

SEE ALSO

chmod(1), rmdir(1), rm(1), umask(2), mkdir(2), rmdir(2)

NAME

mkstr – create an error message file by massaging C source

SYNOPSIS

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr creates files of error messages. You can use *mkstr* to make programs with large numbers of error diagnostics much smaller, and to reduce system overhead in running the program — as the error messages do not have to be constantly swapped in and out.

Mkstr processes each of the specified *files,* placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical example of using *mkstr* would be:

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx.*

To process the error messages in the source to the message file, *mkstr* keys on the string 'error("' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, that is:

```
char    efilname[] = "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

OPTIONS

– Place error messages at the end of the specified message file for recompiling part of a large *mkstr*'d program.

SEE ALSO

lseek(2), xstr(1)

NAME

more, page – browse through a text file

SYNOPSIS

more [**–cdflsu**] [*–lines*] [*+linenumber*] [*+/pattern*] [name ...]

page [**–cdflsu**] [*–lines*] [*+linenumber*] [*+/pattern*] [name ...]

DESCRIPTION

More is a filter which displays the contents of a text file one screenful at a time on a video terminal. It normally pauses after each screenful, and prints '–More–' at the bottom of the screen. *More* displays another line if you type a carriage-return; *more* displays another screenful if you type a space.

If you use the *page* command instead of the *more* command, the screen is cleared before each screenful is displayed (but only if a full screenful is being displayed), and $k-1$ rather than $k-2$ lines are displayed in each screenful, where k is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the –c mode of operation, the *csh* command *"setenv MORE -c"* or the *sh* command sequence *"MORE='-c'; export MORE"* would cause all invocations of *more*, including invocations by programs such as *man* to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.login* or *.profile* file.

If *more* is reading from a file, rather than a pipe, a percentage is displayed along with the –More– prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
: display *i* more lines, (or another screenful if no argument is given)

^D
: display 11 more lines (a "scroll"). If *i* is given, the scroll size is set to *i*.

d
: same as ^D (control-D)

*i*z
: same as typing a space except that *i*, if present, becomes the new window size.

*i*s
: skip *i* lines and print a screenful of lines

*i*f
: skip *i* screenfuls and print a screenful of lines

q or Q
: Exit from *more*.

=
: Display the current line number.

v
: Start up the editor *vi* at the current line.

h
: Help command; give a description of all the *more* commands.

i/expr
: search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found, or the end of the pipe, whichever comes first. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n
: search for the *i*-th occurrence of the last regular expression entered.

'
: (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q
exit from *more* (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately; it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may type the line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can type the quit key (normally control-\). *More* stops sending output, and displays the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

More sets the terminal to *noecho* mode so that the output can be continuous. Thus what you type does not show on your terminal, except for the / and ! commands.

If the standard output is not a terminal, *more* acts just like *cat,* except that a header is printed before each file in a series.

OPTIONS

-lines Set the size of the window to *lines* lines long instead of the default.

-c Display each page by redrawing the screen instead of scrolling. This makes it easier to read text while *more* is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

-d Display the message 'Hit space to continue, Rubout to abort' at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where users are unsophisticated.

-f Count logical rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul,* since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen postions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

-l Do not treat ^L (form feed) specially. If *-l* is not used, *more* pauses after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

-s Squeeze multiple blank lines from the output, and replace them with single blank lines. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

-u Normally, *more* handles underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *-u* option suppresses this processing.

+*linenumber*
Start up at *linenumber*.

+/*pattern*
Start up two lines before the line containing the regular expression *pattern*.

EXAMPLES

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

csh(1), man(1), script(1), sh(1), environ(5), termcap(5)

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**-f** *tapename*] *command* [*count*]

DESCRIPTION

Mt sends commands to a magnetic tape drive. If a tape name is not specified, the environment variable TAPE is used; if TAPE does not exist, *mt* uses the device */dev/rmt12*. Note that *tapename* must reference a raw (not block) tape device. By default *mt* performs the requested operation once. Operations may be performed multiple times by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

eof, weof
Write *count* end-of-file marks at the current position on the tape.

fsf Forward space *count* files.

fsr Forward space *count* records.

bsf Back space *count* files.

bsr Back space *count* records.

rewind
Rewind the tape (*Count* is ignored.)

offline, rewoffl
Rewind the tape and place the tape unit off-line (*Count* is ignored.)

status Print status information about the tape unit.

Mt returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

FILES

/dev/rmt*	Raw magnetic tape interface
/dev/rar*	Raw Archive cartridge tape interface
/dev/rst*	Raw SCSI tape interface

SEE ALSO

mtio(4), dd(1), ioctl(2), environ(5)

BUGS

Archive tapes are not full-fledged tape devices, as they do not support the **fsr, bsf** or **bsr** options.

NAME

mv – move or rename files

SYNOPSIS

mv [**–i**] [**–f**] [–] file1 file2

mv [**–i**] [**–f**] [–] directory1 directory2

mv [**–i**] [**–f**] [–] file ... directory ... directory

DESCRIPTION

Mv moves files and directories around in the file system. A side effect of *mv* is to rename a file or directory. The three major forms of *mv* are shown in the synopsis above.

The first form of *mv* moves (changes the name of) *file1* to *file2*. If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place, otherwise *mv* exits.

The second form of *mv* moves (changes the name of) *directory1* to *directory2*, *ONLY* if *directory2* does not already exist — if it does, the third form applies.

The third form of *mv* moves one or more *files* and *directories*, with their original names, to the last *directory* in the list.

Mv refuses to move a file or directory onto itself.

OPTIONS

–i interactive mode: *mv* displays the name of the file or directory followed by a question mark whenever a move would replace an existing file or directory. If you type a line starting with 'y', *mv* moves the specified file or directory, otherwise *mv* does nothing with that file or directory.

–f force: override any mode restrictions and the **–i** switch. The **–f** option also suppresses any warning messages about modes which would potentially restrict overwriting.

– Interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

SEE ALSO

cp(1), ln(1)

BUGS

If *file1* and *file2* are on different file systems, then *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Mv will not move a directory from one file system to another.

NAME

nice, nohup – run a command at low priority (*sh* only)

SYNOPSIS

nice [*–number*] command [arguments]

nohup command [arguments]

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. '—10'.

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. *Nohup* should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal. The syntax of nice is also different.

FILES

nohup.out standard output and standard error file under *nohup*

SEE ALSO

csh(1), nice(3C), renice(8)

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

Nice and *nohup* are particular to *sh*(1). If you use *csh*(1), then commands executed with "&" are automatically immune to hangup signals while in the background. There is a builtin command *nohup* which provides immunity from terminate, but it does not redirect output to *nohup.out*.

Nice is built into *csh*(1) with a slightly different syntax than described here. The form "nice + 10" nices to positive nice, and "nice –10" can be used by the super-user to give a process more of the processor.

NAME

nm – print name list

SYNOPSIS

nm [–gnoprua] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in *a.out* are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters:

U (undefined),

A (absolute),

T (text segment symbol),

D (data segment symbol),

B (bss segment symbol),

C (common symbol),

f file name,

– debug symbol table entries (see **–a** below).

If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

OPTIONS

–g Print only global (external) symbols.

–n Sort numerically rather than alphabetically.

–o Prepend file or archive element name to each output line rather than only once.

–p Don't sort; print in symbol-table order.

–r Sort in reverse order.

–u Print only undefined symbols.

–a Print all symbols.

EXAMPLE

nm

prints the symbol list of *a.out*, the default output file for the C compiler.

SEE ALSO

ar(1), ar(5), a.out(5)

NAME

nroff – text formatting and typesetting

SYNOPSIS

nroff [**-o***pagelist*] [**-n***N*] [**-s***N*] [**-m***name*] [**-r***aN*] [**-l**] [**-q**] [**-T***name*] [**-e**] [**-h**] [file] ...

DESCRIPTION

Nroff formats text in the named *files* for typewriter-like devices. See also *troff*(1). The full capabilities of *nroff* and *troff* are described in *Formatting Documents with Nroff and Troff*.

If no *file* argument is present, *nroff* reads the standard input. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear *before* the files.

-o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end.

-n*N* Number first generated page *N*.

-s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.

-m*name*
Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

-r*aN* Set register *a* (one-character) to *N*.

-l Read standard input after the input files are exhausted.

-q Invoke the simultaneous input-output mode of the **rd** request.

-T*name*
Prepare output for a device of the specified *name*. Known *names* are:

37	Teletype Corporation Model 37 terminal — this is the default.
tn300	GE TermiNet 300 (or any terminal without half-line capability),
300S	DASI-300S
300	DASI-300
300X	DASI-300X
450	DASI-450 (Diablo Hyterm).
450-12	DASI-450 (Diablo Hyterm) — 12-pitch.
450-12-8	DASI-450 (Diablo Hyterm) — 12-pitch and 8 lines-per-inch.
450X	DASI-450X (Diablo Hyterm).
lpr	Line printer.
x1700	Xerox 1700 daisywheel printer.

-e Produce equally-spaced words in adjusted lines, using full terminal resolution.

-h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

EXAMPLE

```
gaia% nroff -s4 -me users.guide
```

Formats *users.guide* using the **-me** macro package, and stopping every 4 pages.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files

/usr/lib/term/* terminal driving tables for *nroff*

SEE ALSO

Formatting Documents with Nroff and Troff in *Editing and Text Processing on the Sun Workstation.*

troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

NAME

od – octal, decimal, hex, ascii dump

SYNOPSIS

od [-format] [file] [[+]offset[.][b] [label]]

DESCRIPTION

Od displays *file*, or it's standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **-o** (octal) is the default. Dumping continues until end-of-file.

The meanings of the format argument characters are:

a Interpret bytes as characters and display them with their ASCII names. If the **p** character is given also, bytes with even parity are underlined. If the **P** character is given, bytes with odd parity are underlined. Otherwise the parity bit is ignored.

b Interpret bytes as unsigned octal.

c Interpret bytes as ASCII characters. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.

d Interpret (short) words as unsigned decimal.

f Interpret long words as floating point.

h Interpret (short) words as unsigned hexadecimal.

i Interpret (short) words as signed decimal.

l Interpret long words as signed decimal.

o Interpret (short) words as unsigned octal.

s[n] Look for strings of ASCII graphic characters, terminated with a null byte. *N* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.

v Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an "*" in column 1.

w[n] Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

x Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; If "." is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with "**x**" or "**0x**", it is interpreted in hexadecimal. If "**b**" ("**B**") is appended, the offset is interpreted as a block count, where a block is 512 (1024) bytes. If the *file* argument is omitted, an *offset* argument must be preceded by "+".

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

Label will be interpreted as a pseudo-address for the first byte displayed. It will be shown in "()" following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

SEE ALSO

adb(1)

BUGS

A file name argument can't start with "+". A hexadecimal offset can't be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

NAME

pagesize – print system page size

SYNOPSIS

pagesize

DESCRIPTION

Pagesize **prints the size of a page of memory in bytes, as returned by** ***getpagesize*****(2). This program is useful in constructing portable shell scripts.**

SEE ALSO

getpagesize(2)

NAME

passwd – change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the user *name* (your own name by default).

Passwd prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, *UNIX password security*

BUGS

The password file information should be kept in a different data structure allowing indexed access; *dbm*(3X) would probably be suitable.

NAME

pc – Pascal compiler

SYNOPSIS

pc [**-c**] [**-g**] [**-w**] [**-p**] [**-O**] [**-S**] [**-o** *output*] [**-C**] [**-b**] [**-l**] [**-l**] [**-s**] [**-z**]
[**-I**name...] name ...

DESCRIPTION

Pc is a Pascal compiler. If given an argument file ending with *.p*, it compiles the file and loads it into an executable file called, by default, *a.out*.

A program may be separated into more than one *.p* file. *Pc* will compile a number of argument *.p* files into object files (with the extension *.o* in place of .p). Object files may then be loaded into an executable *a.out* file. Exactly one object file must supply a **program** statement to successfully create an executable a.out file. The rest of the files must consist only of declarations which logically nest within the program. References to objects shared between separately compiled files are allowed if the objects are declared in **include**'d header files, whose names must end with *.h*. Header files may only be included at the outermost level, and thus declare only globally available objects. To allow **functions** and **procedures** to be declared, an **external** directive has been added, whose use is similar to the **forward** directive but restricted to appear only in *.h* files. **Function** and **procedure** bodies may not appear in *.h* files. A binding phase of the compiler checks that declarations are used consistently, to enforce the type checking rules of Pascal.

Object files created by other language processors may be loaded together with object files created by *pc*. The **functions** and **procedures** they define must have been declared in *.h* files included by all the *.p* files which call those routines. Calling conventions are as in C, with **var** parameters passed by address.

See the *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual for details.

OPTIONS

The following options have the same meaning as in *cc*(1) and *f77*(1). See *ld*(1) for load-time options.

-c Suppress loading and produce *.o* file(s) from source file(s).

-g Produce additional symbol table information for *dbx*(1).

-w Suppress warning messages.

-p Prepare object files for profiling, see *prof*(1).

-O Invoke an object-code improver.

-S Compile the named program, and leave the assembler-language output on the corresponding file with a *.s* suffix. No *.o* is created.

-o *output*
Name the final output file *output* instead of *a.out*.

The following options are peculiar to *pc*.

-C Compile code to perform runtime checks, verify **assert** statements, and initialize all variables to zero as in *pi*.

-b Block buffer the file *output*.

-l Produce a listing for the specified procedures, functions and **include** files.

-l Make a program listing during translation.

-s Accept standard Pascal only; non-standard constructs cause warning diagnostics.

-z Allow execution profiling with *pxp* by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

Other arguments are taken to be loader option arguments, perhaps libraries of *pc* compatible routines. Certain flags can also be controlled in comments within the program as described in the *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.

FILES

file.p	pascal source files
/usr/lib/pc0	compiler
/lib/f1	code generator
/usr/lib/pc2	runtime integrator (inline expander)
/lib/c2	peephole optimizer
/usr/lib/pc3	separate compilation consistency checker
/usr/lib/pc2.0strings	text of the error messages
/usr/lib/how_pc	basic usage explanation
/usr/lib/libpc.a	intrinsic functions and I/O library
/usr/lib/libm.a	math library
/lib/libc.a	standard library, see *intro*(3)

SEE ALSO

The *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.
pi(1), pxp(1), pxref(1)

DIAGNOSTICS

For a basic explanation do

```
pc
```

BUGS

The **packed** keyword is recognized but has no effect.

The **-s** flag doesn't work for separately compiled files.

Because the **-s** option is usurped by the compiler, it is not possible to pass the strip option to the loader. Thus programs which are to be stripped, must be run through *strip*(1) after they are compiled.

The compiler qualifiles a nested procedure name by chaining the names of the enclosing procedures. This sometimes results in names long enough to abort the assembler (*as*(1)), which currently limits identifiers to 50 characters.

NAME

perfmon – graphical display of general system statistics

SYNOPSIS

perfmon [statistic ...]

DESCRIPTION

Perfmon provides a graphical display of the system-wide performance statistics and updates them approximately once a second. It should be executed from a graphics tool inside the SunWindows system. The time interval between updates can be adjusted by typing one of the following characters while the mouse is in the graphics subwindow:

s increases the interval by 0.05 seconds (small *s* means get *s*lower by a little).

S increases the interval by one second (capital *S* means get *S*lower by a lot).

f decreases the interval by 0.05 seconds (small *f* means get *f*aster by a little).

F decreases the interval by one second (capital *F* means get *F*aster by a lot).

R resets the interval to the standard 1.05 seconds.

In addition, typing:

H, h or ?
lists the characters that *perfmon* is listening for.

Q or q causes *perfmon* to cease executing.

If no statistic argument is given, *perfmon* displays all statistics. A tick is placed on the lines separating the graphs once every fifteen seconds (due to scheduling vagaries, these ticks may not be evenly spaced).

Statistics

user is the percentage of total CPU time spent in normal and low priority user processes.

system
is the percentage of total CPU time attributed to system calls and overhead.

idle is the percentage of total CPU time spent idle.

free is the amount of available real memory (in Kbytes).

disk is the total number of disk transfers performed.

interrupts
is the total number of interrupts serviced.

input is the total number of input packets received.

output
is the total number of output packets transmitted.

collision
is the total number of collisions between packets observed on the network.

SEE ALSO

netstat(8), vmstat(8), suntools(1)

NAME

pi – Pascal interpreter code translator

SYNOPSIS

pi [**-b**] [**-i**] [**-l**] [**-n**] [**-p**] [**-s**] [**-t**] [**-u**] [**-w**] [**-z**] [**-I** name ...] name.p

DESCRIPTION

Pi translates the program in the file *name.p* leaving interpreter code in the file *obj* in the current directory. The interpreter code can be executed using *px*. *Pix* performs the functions of *pi* and *px* for 'load and go' Pascal.

OPTIONS

The following flags are interpreted by *pi;* the associated options can also be controlled in comments within the program as described in the *Pascal User's Manual* in the Sun *Fortran and Pascal Manual.*

-b Block buffer the file *output.*

-i Enable the listing for any specified procedures and functions and while processing any specified **include** files.

-l Make a program listing during translation.

-n Begin each listed **include** file on a new page with a banner line.

-p Suppress the post-mortem control flow backtrace if an error occurs; suppress statement limit counting.

-s Accept standard Pascal only; non-standard constructs cause warning diagnostics.

-t Suppress runtime tests of subrange variables and treat **assert** statements as comments.

-u Card image mode; only the first 72 characters of input lines are used.

-w Suppress warning diagnostics.

-z Allow execution profiling with *pxp* by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

FILES

file.p	input file
file.i	**include** file(s)
/usr/lib/pi2.*strings	text of the error messages
/usr/lib/how_pi*	basic usage explanation
obj	interpreter code output

SEE ALSO

Berkeley Pascal User's Manual
pix(1), px(1), pxp(1), pxref(1)

DIAGNOSTICS

For a basic explanation do

pi

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.

Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow-up messages begin with an ellipsis '...'.

The first character of each error message indicates its class:

E	Fatal error; no code will be generated.
e	Non-fatal error.
w	Warning – a potential problem.
s	Non-standard Pascal construct warning.

If a severe error occurs which inhibits further processing, the translator will give a diagnostic and then 'QUIT'.

BUGS

The keyword **packed** is recognized but has no effect.

For clarity, semantic errors should be flagged at an appropriate place in the source text, and multiple instances of the 'same' semantic error should be summarized at the end of a **procedure** or **function** rather than evoking many diagnostics.

When **include** files are present, diagnostics relating to the last procedure in one file may appear after the beginning of the listing of the next.

NAME

pix – Pascal interpreter and executor

SYNOPSIS

pix [–blnpstuwz] [–l name ...] name.p [argument ...]

DESCRIPTION

Pix is a 'load and go' version of Pascal which combines the functions of the interpreter code translator *pi* and the executor *px*. It uses *pi* to translate the program in the file *name.p* and, if there were no fatal errors during translation, causes the resulting interpreter code to be executed by *px* with the specified arguments. A temporary file is used for the object code; the file *obj* is neither created nor destroyed.

FILES

/usr/bin/pi	Pascal translator
/usr/bin/px	Pascal executor
/tmp/pix*	temporary
/usr/lib/how_pix	basic explanation

SEE ALSO

The *Pascal User's Manual* in the *Fortran and Pascal for the Sun Workstation* Manual.
pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

pix

NAME

plot – graphics filters

SYNOPSIS

plot [–Tterminal [raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(5)) from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter TERM (see *environ*(5)) is used. Known *terminals* are:

4014 Tektronix 4014 storage scope.

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

ver Versatec D1200A printer-plotter. This version of *plot* places a scan-converted image in */usr/tmp/raster* and sends the result directly to the plotter device rather than to the standard output. The optional argument sends a previously scan-converted file called *raster* to the plotter.

FILES

/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/tmp/raster

SEE ALSO

plot(3X), plot(5)

BUGS

There is no lockout protection for */usr/tmp/raster*.

NAME

pmerge – pascal file merger

SYNOPSIS

pmerge name.p ...

DESCRIPTION

Pmerge assembles the named Pascal files into a single standard Pascal program. The resulting program is listed on the standard output. It is intended to be used to merge a collection of separately compiled modules so that they can be run through **pi**, or exported to other sites.

FILES

/usr/tmp/MG* default temporary files

SEE ALSO

pc(1), pi(1),
Auxiliary documentation *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.

BUGS

Very minimal error checking is done, so incorrect programs will produce unpredictable results. Block comments should be placed after the keyword to which they refer or they are likely to end up in bizarre places.

NAME

postnews – submit news articles

SYNOPSIS

postnews [*article*]

DESCRIPTION

Postnews calls up the *inews*(1) utility to submit news articles to USENET. *Postnews* prompts you for the title of the article, for the newsgroup, and for the distribution. The title of the article should be a phrase suggesting the subject, so that persons reading the news can tell if they are interested in the article.

If you omit the name of the newsgroup (by typing a carriage-return when asked for the newsgroup), *postnews* posts the article to *general*.

general is read by everyone on the local machine. Other possible newsgroups include, but are not limited to, **btl.general**, read by all users at all Bell Labs sites on USENET, **net.general**, read by all users at all sites on USENET, and **net.news**, read by users interested in the network news on all sites. There is often a local set of newsgroups, such as **ucb.all**, that circulate within a local set of machines. In this case, *ucb* newsgroups circulate among machines at the University of California at Berkeley.

The distribution can be any valid list of newsgroup names, and defaults to the same as the newsgroup. If they are the same, the distribution is omitted from the headers put into the editor buffer. A distribution header is included in the headers of the article if given, affecting where the article is distributed to.

After entering the title, newsgroup, and distribution, *postnews* calls up an editor program where you can compose the article. *Postnews* uses *vi*(1) as the editor unless you have specified another editor via the $EDITOR environment variable, in which case *postnews* uses the editor specified there.

An initial set of headers containing the subject and newsgroups will be placed in the editor, followed by a blank line. The article should be appended to the buffer, after the blank line. These headers can be changed, or additional headers added, while in the editor, if desired.

Optionally, the article is read from the file specified by *article*.

For more sophisticated uses, such as posting news from a program, see *inews*(1).

SEE ALSO

mail(1), checknews(1), inews(1), readnews(1).

NAME

pr – print file(s), possibly in multiple columns

SYNOPSIS

pr [**–***n*] [**+***n*] [**–h** *string*] [**–w***n*] [**–f**] [**–l***n*] [**–t**] [**–s***n*] [**–m**] [file] ...

DESCRIPTION

Pr prepares one or more *files*'s for printing. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. *Pr* prints its standard input if there are no *file* arguments.

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

OPTIONS

Options apply to all following *file*'s but may be reset between *file*'s:

–*n* Produce *n*-column output. This option overrides the **–t** option (see below).

+*n* Begin printing with page *n*.

–h *string*
Use *string* as a header for the page instead of the default header.

–w*n* For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.

–f Use formfeeds instead of newlines to separate pages. A formfeed is assumed to use up two blank lines at the top of a page. Thus this option does not affect the effective page length.

–l*n* Take the length of the page to be *n* lines instead of the default 66.

–t Do not print the 5-line header or the 5-line trailer normally supplied for each page. Formfeed characters are not generated when this option is used, even if the **–f** option was used. The **–t** option is intended for applications where the results should be directed to a file for further processing.

–s*c* Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.

–m Print all *file*'s simultaneously, each in one column,

EXAMPLES

Print a file called *dreadnaught* on the printer — this is the simplest use of *pr*:

```
krypton% pr dreadnaught | lpr
krypton%
```

Produce three laminations of a file called *ridings* side by side in the output, with no headers or trailers, the results to appear in the file called *Yorkshire*:

```
krypton% pr -m -t ridings ridings ridings > Yorkshire
krypton%
```

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(1), lpr(1)

DIAGNOSTICS

There are no diagnostics when *pr* is printing on a terminal.

BUGS

The options described above interact with each other in strange and as yet to be defined ways.

NAME

printenv – print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv **prints out the values of the variables in the environment. If a** ***name*** **is specified, only its value is printed.**

If a ***name*** **is specified and it is not defined in the environment,** ***printenv*** **returns exit status 1, else it returns status 0.**

SEE ALSO

sh(1), environ(5), csh(1)

NAME

prmail – print out waiting mail

SYNOPSIS

prmail [user ...]

DESCRIPTION

Prmail prints the mail which waits for you, or the specified *users*. The mail is not disturbed.

FILES

/usr/spool/mail/* waiting mail files

SEE ALSO

biff(1), mail(1), from(1), binmail(1)

NAME

prof – display profile data

SYNOPSIS

prof [**-a**] [**-l**] [**-n**] [**-s**] [**-z**] [**-v** [*-low* [*-high*]]] [a.out [mon.out ...]]

DESCRIPTION

Prof interprets the file produced by the *monitor* subroutine. Under default modes, the symbol table in the named object file (*a.out* by default) is read and correlated with the profile file (*mon.out* default). For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call. If more than one profile file is specified, the output represents the sum of the profiles.

To tally the number of calls to a routine, the program must be compiled with the **-p** option of *cc*, *f77* or *pc*. This option also means that the profile file is produced automatically.

OPTIONS

-a Report all symbols rather than just external symbols.

-l Sort the output by symbol value.

-n sort the output by number of calls.

-s Produce a summary profile file in *mon.sum*. This is really only useful when more than one profile file is specified.

-v Suppress all printing and produce a graphic version of the profile on the standard output for display by the *plot*(1G) filters. When plotting, the numbers *low* and *high*, (by default 0 and 100), select a percentage of the profile to be plotted with accordingly higher resolution.

-z Print routines which have zero usage (as indicated by call counts and accumulated time).

FILES

mon.out for profile
a.out for namelist
mon.sum for summary profile

SEE ALSO

monitor(3), profil(2), cc(1), plot(1G), gprof(1)

BUGS

Beware of quantization errors.

Is confused by *f77* which puts the entry points at the bottom of subroutines and functions.

NAME

prs – print an SCCS file

SYNOPSIS

/usr/sccs/prs [**–d**[*dataspec*]] [**–r**[*SID*]] [**–e**] [**–l**] [**–a**] file ...

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of – is given, the standard input is read, in which case each line is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

OPTIONS

Options apply independently to each named file.

–d[*dataspec*]
: Specifies the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

–r[*SID*]
: Specifies the *S*CCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

–e
: Requests information for all deltas created *earlier* than and including the delta designated via the **–r** option.

–l
: Requests information for all deltas created *later* than and including the delta designated via the **–r** option.

–a
: Requests printing of information for both removed, that is, delta type = *R*, (see *rmdel*(1)) and existing, that is, delta type = *D*, deltas. If the **–a** option is not specified, information for existing deltas only is provided.

In the absence of the **–d** options, *prs* displays a default set of infomation consisting of: delta-type, release number and level number, date and time last changed, user-name of the person who changed the file, lines inserted, changed, and unchanged, the MR numbers, and the comments.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: 1) the user supplied text; and 2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by **\t** and carriage return/new-line is specified by **\n**.

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th::Tm::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

/usr/sccs/prs –d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:
xyz
131
abc

/usr/sccs/prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case:*

/usr/sccs/prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only option argument allowed to be used with the *special case* is the **–a** option.

FILES

/tmp/pr?????

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – process status

SYNOPSIS

ps [acegklstuvwx#]

DESCRIPTION

Ps displays information about processes. Normally, only processes that you have started are candidates to be displayed by *ps*, but see the OPTIONS section below for how to get more information. Specifying **a** makes other users' processes candidates to be displayed; specifying **x** includes processes without control terminals in the candidate pool.

All output formats include, for each process, the process id PID, control terminal of the process TT, cpu time used by the process TIME (this includes both user and system time), the state STAT of the process, and an indication of the COMMAND which is running.

The state is given by a sequence of four letters, for example, 'RWNA'.

First letter

indicates the runnability of the process:

R Runnable processes,
T Stopped processes,
P Processes in page wait,
D Processes in disk (or other short term) waits,
S Processes sleeping for less than about 20 seconds,
I Processes which are idle (sleeping longer than about 20 seconds).

Second letter

indicates whether a process is swapped out;

blank
(that is, a space) in this position indicates that the process is loaded (in memory).

W Process is swapped out.

> Process has specified a soft limit on memory requirements and has exceeded that limit shows; such a process is (necessarily) not swapped.

Third letter

indicates whether a process is running with altered CPU scheduling priority (nice):

blank
(that is, a space) in this position indicates that the process is running without special treatment.

N The process priority is reduced,

< The process priority has been raised artificially.

Fourth letter

indicates any special treatment of the process for virtual memory replacement. The letters correspond to options to the *vadvise*(2) system call. Currently the possibilities are:

blank
(that is, a space) in this position stands for VA_NORM.

A Stands for VA_ANOM. An A typically represents a program which is doing garbage collection.

S Stands for VA_SEQL. An S is typical of large image processing programs which are using virtual memory to sequentially address voluminous data.

OPTIONS

a Display information about all processes with terminals (ordinarily only one's own processes are displayed).

e 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.

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 Use the file */vmcore* in place of */dev/kmem* and */dev/mem*. This is used for postmortem system debugging.

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.

t*x* Restrict output to processes whose controlling tty is *x* (which should be specified as printed by *ps*, for example, *t3* for tty3, *tco* for console, *td0* for ttyd0, *t?* for processes with no tty, etc). This option must be the last one given.

u Display user-oriented output. This includes fields USER, %CPU, NICE, SIZE, and RSS as described below.

v Display a version of the output containing virtual memory. This includes fields RE, SL, PAGEIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU and %MEM, described below.

w Use a wide output format (132 columns rather than 80); if repeated, that is, ww, use arbitrarily wide output. This information is used to decide how much of long commands to print.

x Display even those processes with no terminal.

A process number may be given, (indicated here by #), in which case the output is restricted to that process. This option must also be last.

A second argument tells *ps* where to look for *core* if the **k** option is given, instead of /vmcore. A third argument is the name of a swap file to use instead of the default /dev/drum. If a fourth argument is given, it is taken to be the file containing the system's namelist. Otherwise, /vmunix is used.

DISPLAY FORMATS

Fields which are not common to all output formats:

USER	name of the owner of the process
%CPU	cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all %CPU fields to exceed 100%.
NICE	(or NI) process scheduling increment (see *setpriority*(2) and *nice*(3C).
SIZE	virtual size of the process (in 1024 byte units)
RSS	real memory (resident set) size of the process (in 1024 byte units)
LIM	soft limit on memory used, specified via a call to *getrlimit*(2); if no limit has been specified then shown as *xx*
TSIZ	size of text (shared program) image
TRS	size of resident (real memory) set of text
%MEM	percentage of real memory used by this process.
RE	residency time of the process (seconds in core)
SL	sleep time of the process (seconds blocked)
PAGEIN	number of disk i/o's resulting from references by the process to pages not loaded in core.
UID	numerical user-id of process owner
PPID	numerical id of parent of process

CP short-term cpu utilization factor (used in scheduling)
PRI process priority (non-positive when in non-interruptible wait)
ADDR swap address of the process
WCHAN event on which process is waiting (an address in the system), with the initial part of the address trimmed off, for example, 80004000 prints as 4000.

F flags associated with process as in <*sys/proc.h*>:

SLOAD	0000001	in core
SSYS	0000002	swapper or pager process
SLOCK	0000004	process being swapped out
SSWAP	0000008	save area flag
STRC	0000010	process is being traced
SWTED	0000020	another tracing flag
SULOCK	0000040	user settable lock in core
SPAGE	0000080	process in page wait state
SKEEP	0000100	another flag to prevent swap out
.		
SOMASK	0000200	restore old mask after taking signal
SWEXIT	0000400	working on exiting
SPHYSIO	0000800	doing physical i/o (bio.c)
SVFORK	0001000	process resulted from vfork()
SVFDONE	0002000	another vfork flag
SNOVM	0004000	no vm, parent in a vfork()
SPAGI	0008000	init data space on demand, from inode
.		
SSEQL	0010000	user warned of sequential vm behavior
SUANOM	0020000	user warned of anomalous vm behavior
STIMO	0040000	timing out during sleep
.		
SOUSIG	0100000	using old signal mechanism
SOWEUPC	0200000	owe process an addupc() call at next ast
SSEL	0400000	selecting; wakeup/waiting danger
SLOGIN	0800000	a login process (legit child of init)
SPTECHG	1000000	pte's for process have changed

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>; a process which is blocked trying to exit is marked <exiting>; *ps* makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory
/dev/drum	swap device
/vmcore	core file
/dev	searched to find swap device and tty names

SEE ALSO

kill(1), w(1)

BUGS

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

NAME

pti – phototypesetter interpreter

SYNOPSIS

pti [file ...]

DESCRIPTION

Pti shows the commands in a stream from the standard output of *troff*(1) using *troff's* –t option, interpreting them as they would act on the typesetter. Horizontal motions shows as counts in internal units and are marked with '<' and '>' indicating left and right motion. Vertical space is called *leading* and is also indicated.

The output is really cryptic unless you are an experienced C/A/T hardware person. It is better to use **troff** –**a**.

SEE ALSO

troff(1)

NAME

ptx – permuted index

SYNOPSIS

ptx [**–f**] [**–t**] [**–w** *n*] [**–g** *n*] [**–o** *only*] [**–l** *ignore*] [**–b** *break*] [**–r**] [input [output]]

DESCRIPTION

Ptx generates a permuted index of the contents of file *input* onto file *output* (defaults are standard input and output). *Ptx* has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an *nroff*(1) or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

OPTIONS

–f Fold upper and lower case letters for sorting.

–t Prepare the output for the phototypesetter; the default line length is 100 characters.

–w*n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

–g*n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

–o*only* Use as keywords only the words given in the *only* file.

–l*ignore*

Do not use as keywords any words given in the *ignore* file. If the **–l** and **–o** options are missing, use */usr/lib/eign* as the *ignore* file.

–b*break*

Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

–r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

FILES

/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

pwd – print working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

If you are using *csh*(1), you can use the *dirs* builtin command to do the same job more quickly; *BUT dirs* can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it. This is because *pwd* searches back up the directory tree to report the true pathname, whereas *dirs* remembers the pathname from the last *cd* command. The example below illustrates the differences.

```
% cd /usr/wendy/january/reports
% pwd
/usr/wendy/january/reports
% dirs
~/january/reports
% mv ~/january ~/february
% pwd
/usr/wendy/february/reports
% dirs
~/january/reports
%
```

Pwd and *dirs* also give different answers when you change directory through a symbolic link. For example:

```
% cd /usr/wendy/january/reports
% pwd
/usr/wendy/january/reports
% dirs
~/january/reports
% ls -l /usr/wendy/january
lrwxrwxrwx 1 wendy 17 Jan 30 1983 /usr/wendy/january -> /usr/wendy/1984/jan/
% cd /usr/wendy/january
% pwd
/usr/wendy/1984/jan
% dirs
/usr/wendy/january
```

SEE ALSO

cd(1), csh(1), getwd(3)

NAME

px – Pascal interpreter

SYNOPSIS

px [obj [argument ...]]

DESCRIPTION

Px interprets the abstract machine code generated by *pi*. The first argument is the file to be interpreted, and defaults to *obj*; remaining arguments are available to the Pascal program using the built-ins *argv* and *argc*. *Px* is also invoked by *pix* when running 'load and go'.

If the program terminates abnormally an error message and a control flow backtrace are printed. The number of statements executed and total execution time are printed after normal termination. The **p** option of *pi* suppresses all of this except the message indicating the cause of abnormal termination.

FILES

obj	default object file
pmon.out	profile data file

SEE ALSO

The *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.
pi(1), pix(1)

DIAGNOSTICS

Most run-time error messages are self-explanatory. Some of the more unusual ones are:

Reference to an inactive file
: A file other than *input* or *output* was used before a call to *reset* or *rewrite*.

Statement count limit exceeded
: The limit of 500,000 executed statements (which prevents excessive looping or recursion) has been exceeded.

Bad data found on integer read
Bad data found on real read
: Usually, non-numeric input was found for a number. For reals, Pascal requires digits before and after the decimal point so that numbers like '.1' or '21.' evoke the second diagnostic.

panic: *Some message*
: Indicates a internal inconsistency detected in *px* probably due to a Pascal system bug.

BUGS

Post-mortem traceback is not limited; infinite recursion leads to almost infinite traceback.

NAME

pxp – Pascal execution profiler

SYNOPSIS

pxp [-acdefjnstuw_] [-23456789] [-z [name ...]] name.p

DESCRIPTION

Pxp can be used to obtain execution profiles of Pascal programs or as a pretty-printer. To produce an execution profile all that is necessary is to translate the program specifying the **z** option to *pi* or *pix*, execute the program, and then type the command

tutorial% **pxp -z name.p**

Pxp generates a reformatted listing if none of the **c**, **t**, or **z** options are specified; thus

tutorial% **pxp old.p > new.p**

places a pretty-printed version of the program in *old.p* in the file *new.p*.

OPTIONS

The use of the following options of *pxp* is discussed in the *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.

-a Print the bodies of all procedures and functions in the profile; even those which were never executed.

-c Extract profile data from the file *core*.

-d Include declaration parts in a profile.

-e Eliminate **include** directives when reformatting a file; the **include** is replaced by the reformatted contents of the specified file.

-f Fully parenthesize expressions.

-j Left justify all procedures and functions.

-n Eject a new page as each file is included; in profiles, print a blank line at the top of the page.

-s Strip comments from the input text.

-t Print a table summarizing **procedure** and **function** call counts.

-u Card image mode; only the first 72 characters of input lines are used.

-w Suppress warning diagnostics.

-z Generate an execution profile. If no *names* are given the profile is of the entire program. If a list of *names* is given, then only any specified **procedures** or **functions** and the contents of any specified **include** files will appear in the profile.

-_ Underline keywords.

-*d* use *d* spaces (where *d* is a digit, $2 \leq d \leq 9$) as the basic indenting unit. The default is 4.

FILES

name.p	input file
name.i	include file(s)
pmon.out	profile data
core	profile data source with -c
/usr/lib/how_pxp	information on basic usage

SEE ALSO

The *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.
pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

pxp

Error diagnostics include 'No profile data in file' with the **c** option if the **z** option was not enabled to *pi;* 'Not a Pascal system core file' if the core is not from a *px* execution; 'Program and count data do not correspond' if the program was changed after compilation, before profiling; or if the wrong program is specified.

BUGS

Does not place multiple statements per line.

NAME

pxref – Pascal cross-reference program

SYNOPSIS

pxref [–] name

DESCRIPTION

Pxref makes a line numbered listing and a cross-reference of identifier usage for the program in *name*. The optional '–' argument suppresses the listing. The keywords **goto** and **label** are treated as identifiers for the purpose of the cross-reference. **Include** directives are not processed, but cause the placement of an entry indexed by '#include' in the cross-reference.

SEE ALSO

The *Pascal User's Manual* in the Sun *Fortran and Pascal* Manual.

BUGS

Identifiers are trimmed to 10 characters.

NAME

ranlib – convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

Ranlib converts each *archive* to a form which the loader can load more rapidly. *Ranlib* does this by adding a table of contents called **_.SYMDEF** to the beginning of the archive. *Ranlib* uses *ar*(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system which contains the current directory.

SEE ALSO

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

BUGS

Because generation of a library by *ar* and randomization of the library by *ranlib* are separate processes, phase errors are possible. The loader, *ld*, warns when the modification date of a library is more recent than the creation date of its dictionary; but this means that you get the warning even if you only copy the library.

NAME

ratfor – rational Fortran dialect

SYNOPSIS

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

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:

{ statement; statement; statement }

decision-making:

if (condition) statement [else statement]
switch (integer value) {
case integer: statement
...
[default:] statement
}

loops:

while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [until (condition)]
break
next

and some syntactic sugar to make programs easier to read and write:

free form input:

multiple statements/line; automatic continuation

comments:

this is a comment

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return(expression)

returns expression to caller from function

define: define name replacement

include:

include filename

Ratfor is best used with *f77*(1).

OPTIONS

–6*c* Use the character *c* as the continuation character in column **6** when translating to Fortran. The default is to use the **&** character as a continuation character.

–C Pass Ratfor comments through to the translated code.

–h Translate Ratfor string constants to Hollerith constants of the form *nnn***h***string*. Otherwise just pass the strings through to the translated code.

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

rcp – remote file copy

SYNOPSIS

rcp file1 file2
rcp [–r] file ... directory

DESCRIPTION

Rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form "rhost:path", or a local file name (containing no ':' characters, or a '/' before any ':'s.)

If the **–r** is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

Rcp does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via *rsh*(1C).

Rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form "rhost.rname" to use *rname* rather than the current user name on the remote host.

Please note: *rcp* is meant to copy from one host to another; if by some chance you try to copy a file on top of itself, you will end up with a severely corrupted file (for example, if you executed the following command from host george: 'george% rcp testfile george:/usr/me/testfile'). Remember where you are at all times (putting your hostname in your prompt helps with this)!

SEE ALSO

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

BUGS

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

Is confused by any output generated by commands in a .login, .profile, or .cshrc file on the remote host.

Rcp doesn't copy ownership, mode, and timestamps to the new files.

NAME

readnews – read news articles

SYNOPSIS

readnews [**–a** *date*] [**–n** *newsgroups*] [**–t** *titles*] [**–lprxhfuM**] [**–c** [*mailer*]]

readnews –s

DESCRIPTION

Readnews without argument displays unread articles.

readnews –s displays the newsgroup subscription list.

Readnews maintains a *.newsrc* file in your home directory that specifies all news articles already read. *.newsrc* is updated at the end of each news reading session in which the **–x** or **–l** options weren't specified. If the NEWSRC environment variable is present, it should be the path name of a file to be used in place of *.newsrc*.

An options line may be placed in the *.newsrc* file. The options line starts with the word **options** (left justified) followed by the list of standard options just as they would be typed on the *readnews* command line. The list of options may include: the **–n** flag along with a newsgroup list; a favorite interface to use for reading the news; and/or the **–r** or **–t** flag. Continuation lines are specified by following lines beginning with a space or tab character. Similarly, options can be specified in the NEWSOPTS environment parameter. Options on the command line override options in the *.newsrc* file and options in the *.newsrc* file override options in the NEWSOPTS environment parameter.

When you use the reply command of the *mail*(1) or */bin/mail*(1) interfaces, *readnews* uses the MAILER environment parameter to determine which mailer to use. The default is usually *mail*.

You can specify a particular paging program for paging through articles. The PAGER environment parameter should be set to the name of the paging program. The name of the article is referenced with a '%', as in the **–c** option. If no '%' is present, the article is piped to the program. Paging may be disabled by setting PAGER to a null value.

OPTIONS

Some of the option flags determine which of the several interfaces you can use for reading your news. The news system has its own interface which is used if no other choice is made, otherwise one of these options can be used:

–M An interface to *mail*(1).

–c A */bin/mail*(1)–like interface.

–c *'mailer'*

All selected articles written to a temporary file. Then the mailer is invoked. The name of the temporary file is referenced with a '%'. Thus, 'mail –f %' will invoke mail on a temporary file consisting of all selected messages.

Other options govern the behavior of *readnews* itself, as follows:

–p Send all selected articles to the standard output with no questions asked.

–l Display only the titles. Do not update the *.newsrc* file.

–r display the articles in reverse order.

–f Do not display any followup article

–h Display articles in a less verbose format (intended for terminals running at 300 baud).

–u Update the *.newsrc* file every 5 minutes, in case of an unreliable system. Note that if the *.newsrc* file is updated, the **x** command will not restore it to its original contents.

The following flags determine the selection of articles:

-n *newsgroups*
Select all articles that belong to *newsgroups*.

-t *titles*
Select all articles whose titles contain one of the strings specified by *titles*.

-a [*date*]
Select all articles that were posted past the given *date* (in *getdate*(3) format).

-x Ignore *.newsrc* file. That is, select articles that have already been read as well as new ones.

COMMANDS

This section lists the commands you can type to the *readnews* and */bin/mail* interface prompts. The *readnews* interface suggests some common commands in brackets. Just typing carriage-return is the same as typing the first command. For example, '[ynq]' means that the commands 'y' (yes), 'n' (no), and 'q' (quit) are common responses, and that 'y' is the default. Here are the commands and their meanings:

y Yes — displays current article and goes on to next.

n No — goes on to next article without displaying current one. In the */bin/mail* interface, this means 'go on to the next article', which has the same effect as 'y' or just typing carriage-return.

q Quit — the *.newsrc* file is updated if -l or -x were not on the command line.

c Cancel the article — only the author or the super user can do this.

r Reply — reply to article's author via mail. *Readnews* calls up your EDITOR with a header specifying To, Subject, and References lines taken from the message. You may change or add headers, as appropriate. You add the text of the reply after the blank line, and then exit the editor. The resulting message is mailed to the author of the article.

rd Reply directly — *readnews* calls up the *mail* program (or the program specified in the $MAILER environment variable) so that you can reply to the author. Type the text of the reply and then control-D.

f [*title*] Submit a follow up article. Normally you should leave off the title, since the system generates one for you. *Readnews* calls up your EDITOR so that you can compose the text of the followup.

fd Followup directly, without edited headers. This is like *f*, but the headers of the article are not included in the editor buffer.

N [*newsgroup*]
Go to the next newsgroup or named *newsgroup*.

s [*file*] Save — the article is appended to the named *file*. The default filename is *Articles*. If the first character of the filename is '|', the rest of the filename is taken as the name of a program, which is executed with the text of the article as standard input. If the first character of the filename is '/', it is taken as a full path name of a file. If the $NEWSBOX environment variable is set to a full path name, and the filename contains no '/', the file is saved in $NEWSBOX, otherwise, it is saved relative to $HOME.

Report the name and size of the newsgroup.

e Erase — forget that this article was read.

h Print a more verbose header.

H Print a very verbose header, containing all known information about the article.

U Unsubscribe from this newsgroup. Also goes on to the next newsgroup.

d Read a digest. Breaks up a digest into separate articles so you can read and reply to each piece.

D Decrypt — invokes a Caesar decoding program on the body of the message. This is used to decrypt rotated jokes posted to net.jokes. Such jokes are usually obscene or otherwise offensive to some groups of people, and so are rotated to avoid accidental decryption by people who would be offended. The title of the joke should indicate the nature of the problem, enabling people to decide whether to decrypt it or not.

Normally the Caesar program does a character frequency count on each line of the article separately, so that lines which are not rotated will be shown in plain text. This works well unless the line is short, in which case it sometimes gets the wrong rotation. An explicit *number* rotation (usually 13) may be given to force a particular shift.

v Print the current version of the news software.

! Shell escape.

number Go to *number*.

+ [*n*] Skip n articles. The articles skipped are recorded as 'unread' and will be offered to you again the next time you read news.

– Go back to last article. This is a toggle, typing it twice returns you to the original article.

x Exit — like quit except that *.newsrc* is not updated.

X *system*
Transmit article to the named *system*.

The c, f, fd, r, rd, e, h, H, and s commands can be followed by –'s to refer to the previous article. Thus, when replying to an article using the *readnews* interface, you should normally type 'r–' (or 're-') since by the time you enter a command, you are being offered the next article.

EXAMPLES

readnews
Read all unread articles using the *readnews*(1) interface. The *.newsrc* file is updated at the end of the session.

readnews –c ed % –l
Use the *ed*(1) text editor on a file containing the titles of all unread articles. The *.newsrc* file is not updated at the end of the session.

readnews –n all !fa.all –M –r
Read all unread articles except articles whose newsgroups begin with IB fa . via *mail*(1) in reverse order. The *.newsrc* file is updated at the end of the session.

readnews –p –n all –a last thursday
Print every unread article since last Thursday. The *.newsrc* file is updated at the end of the session.

readnews –p > /dev/null &
Discard all unread news. This is useful after returning from a long trip.

FILES

/usr/spool/news/*newsgroup*/*number*	News articles
/usr/lib/news/active	Active newsgroups and numbers of articles
/usr/lib/news/help	Help file for *readnews*(1) interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

checknews(1), inews(1), sendnews(8), recnews(8), uurec(8), mail(1), news(5), newsrc(5)

NAME

recnews – receive unprocessed articles via mail

SYNOPSIS

/usr/lib/news/recnews [*newsgroup* [*sender*]]

DESCRIPTION

Recnews reads a letter from the standard input; determines the article title, sender, and newsgroup; and gives the body to inews with the right arguments for insertion.

If *newsgroup* is omitted, the to line of the letter is used. If *sender* is omitted, the sender is determined from the from line of the letter. The title is determined from the subject line.

SEE ALSO

inews(1), uurec(8), sendnews(8), readnews(1), checknews(1)

NAME

refer – find and insert literature references in documents

SYNOPSIS

refer [**-a***r*] [**-b**] [**-c***string*] [**-e**] [**-k***x*] [**-l***m,n*] [**-p** *file*] [**-n**] [**-s***keys*] file ...

DESCRIPTION

Refer is a preprocessor for *nroff*(1), or *troff*(1), that finds and formats references. The input files (standard input by default) are copied to the standard output, except for lines between .[and .] command lines. Such lines are assumed to contain keywords as for *lookbib*(1), and are replaced by information from a bibliographic data base. The user can avoid the search, override fields from it, or add new fields. The reference data, from whatever source, is assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. A flag is placed in the text at the point of reference. By default, the references are indicated by numbers.

When *refer* is used with *eqn*(1), *neqn*(1), or *tbl*(1), *refer* should be used first in the sequence, to minimize the volume of data passed through pipes.

OPTIONS

-a*r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted, all author names are reversed.

-b Bare mode — do not put any flags in text (neither numbers or labels).

-c*string*
Capitalize (with SMALL CAPS) the fields whose key-letters are in *string*.

-e Accumulate references instead of leaving the references where encountered, until a sequence of the form:

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

-k*x* Instead of numbering references, use labels as specified in a reference data line beginning with the characters *%x*; By default, *x* is **L**.

-l*m,n* Instead of numbering references, use labels from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either of *m* or *n* is omitted, the entire name or date, respectively, is used.

-p Take the next argument as a file of references to be searched. The default file is searched last.

-n Do not search the default file.

-s*keys* Sort references by fields whose key-letters are in the *keys* string, and permute reference numbers in the text accordingly. Using this option implies the **-e** option. The key-letters in *keys* may be followed by a number indicating how many such fields are used, with a + sign taken as a very large number. The default is **AD**, which sorts on the senior author and date. To sort on all authors and then the date, for instance, use the options **-sA+T**.

FILES

/usr/dict/papers	directory of default publication lists and indexes
/usr/lib/refer	directory of programs

NAME

reset – reset the teletype bits to a sensible state

SYNOPSIS

reset

DESCRIPTION

Reset sets the terminal to cooked mode, turns off cbreak and raw modes, turns on nl, and restores special characters that are undefined to their default values.

This is most useful after a program dies leaving a terminal in a funny state; you have to type "<LF>reset<LF>" to get it to work then to the shell, as <CR> often doesn't work; often none of this will echo.

It's a good idea to follow *reset* with *tset*(1)

SEE ALSO

stty(1), tset(1)

BUGS

Doesn't set tabs properly; it can't intuit personal choices for interrupt and line kill characters, so it leaves these the standard settings which are ^H (backspace) for erase character, ^U for line kill character, and ^C for interrupt character.

It could well be argued that the shell should be responsible for ensuring that the terminal remains in a sane state; this would eliminate the need for this program.

NAME

rev – reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

NAME

rlogin – remote login

SYNOPSIS

rlogin *rhost* [**–e** *c*] [**–l** *username*] [**–8**]
rhost [**–e** *c*] [**–l** *username*] [**–8**]

DESCRIPTION

Rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rsh*(1C).) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file *.rhosts* in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user and host is not found in these files, the remote machine will prompt for a password as in *login*(1). To avoid some security problems, the *.rhosts* file must be owned by either the remote user or root and may not be a symbolic link.

Your remote terminal type is the same as your local terminal type (as given in your environment TERM variable). All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly.

ESCAPES

Lines starting with the tilde character are 'escape sequences' (the escape character can be changed via the **–e** options):

~. Disconnect from the remote host — this is not the same as a logout, because the local host breaks the connection with no warning to the remote end.

~^Z Suspend the login session (only if you are using the C-Shell). *Susp* is your 'suspend' character — usually control-Z — see *tty*(1).

~^Y Suspend the input half of the login, but output will still be seen (only if you are using the C-Shell). *Dsusp* is your 'deferred suspend' character — usually control-Y — see *tty*(1).

OPTIONS

–l Specifies a different user name (*username*, in the synopsis) for the remote login. If you do not use this option, the remote username used is the same as your local username.

–e Specifies a different escape character (*c*, in the synopsis) for the line used to disconnect from the remote host.

–8 Pass eight-bit data across the net instead of seven-bit data.

SEE ALSO

rsh(1C), stty(1)

FILES

/usr/hosts/*	for *rhost* version of the command
/etc/hosts	to translate hostname to network address
/etc/hosts.equiv	list of *rhost*s with shared account names
~/.rhosts	private list of *rhost*s with shared account names
/etc/services	to translate service name tcp/rlogin to network port number

BUGS

This implementation can only use the TCP network service.

More terminal characteristics should be propagated.

NAME

rm, rmdir – remove (unlink) files or directories

SYNOPSIS

rm [**–f**] [**–r**] [**–i**] [**–**] file ...

rmdir dir ...

DESCRIPTION

Rm removes the directory entries for one or more files. If an entry was the last link to the file, the file is destroyed. *Rm* **–r** and *rmdir* remove entries for directories.

To remove a file, you must have write permission in its directory; but you don't need read or write permission on the file itself. If you don't have write permission on the file and the standard input is a terminal, *rm* displays the file's permissions and waits for you to type in a response. If your response begins with 'y' the file is deleted; otherwise the file is left alone.

To remove a full directory, use *rm* with the **–r** option (see below). *Rmdir* removes the named directory only if it is empty.

OPTIONS

The following are options for *rm*:

–f Force files to be removed, without displaying permissions, asking questions, or reporting errors.

–r Recursively delete the entire contents of the specified directory and the directory itself.

–i Ask whether to delete each file, or, under **–r**, whether to examine each directory. Sometimes called the interactive option.

– Treat all the following arguments as filenames — so that you can specify filenames starting with a minus.

WARNING

It is forbidden to remove the file '..' merely to avoid the antisocial consequences of inadvertently doing something like 'rm –r .*'.

SEE ALSO

rmdir(1), rmdir(2), unlink(2)

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

/usr/sccs/rmdel **–r***SID* **file ...**

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS *file*. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (that is, if a *p-file* (see *get*(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*).

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

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either 1) if you make a delta you can remove it; or 2) if you own the file and directory you can remove a delta.

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

rmdir, rm – remove (unlink) directories or files

SYNOPSIS

rmdir dir ...

rm [**–f**] [**–r**] [**–i**] [**–**] file ...

DESCRIPTION

Rmdir removes entries for the named directories, which must be empty.

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked and no errors are reported when the **–f** (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument **–r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **–i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **–r**, whether to examine each directory.

The null option **–** indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

SEE ALSO

rm(1), unlink(2), rmdir(2)

NAME

roffbib – run off bibliographic database

SYNOPSIS

roffbib [–e –h –n –o –r –s –Tterm –x –m mac –V –Q] [file ...]

DESCRIPTION

Roffbib prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with *sortbib:*

```
sortbib database | roffbib
```

Roffbib accepts most of the options understood by *nroff*(1), most importantly the –T flag to specify terminal type.

If abstracts or comments are entered following the %X field key, *roffbib* will format them into paragraphs for an annotated bibliography. Several %X fields may be given if several annotation paragraphs are desired. The –x flag will suppress the printing of these abstracts.

A user-defined set of macros may be specified after the –m option. There should be a space between the –m and the macro filename. This set of macros will replace the ones defined in /usr/lib/tmac/tmac.bib. The –V flag will send output to the Versatec; the –Q flag will queue output for the phototypesetter.

Four command-line registers control formatting style of the bibliography, much like the number registers of *ms*(7). The command-line argument –rN1 will number the references starting at 1. The flag –rV2 will double space the bibliography, while –rV1 will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the –rL6i argument, and the page offset can be set from the default of 0 to one inch by specifying –rO1i (capital O, not zero). Note: with the –V and –Q flags the default page offset is already one inch.

FILES

/usr/lib/tmac/tmac.bib file of macros used by *nroff/troff*

SEE ALSO

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

BUGS

Users have to rewrite macros to create customized formats.

NAME

rsh – remote shell

SYNOPSIS

rsh host [–l username] [–n] command
host [–l username] [–n] command

DESCRIPTION

Rsh connects to the specified *host,* and executes the specified *command. Rsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does.

If you omit *command,* instead of executing a single command, *rsh* logs you in on the remote host using *rlogin*(1C).

Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command:

tutorial% **rsh lizard cat lizard.file >> tutorial.file**

appends the remote file *lizard.file* from the machine called *lizard* to the file called *tutorial.file* on the machine called *tutorial,* while the command:

tutorial% **rsh lizard cat lizard.file ">>" another.lizard.file**

appends the file *lizard.file* on the machine called *lizard* to the file *another.lizard.file.* which also resides on the machine called *lizard.*

Host names are given in the file /etc/hosts. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory /usr/hosts; if you put this directory in your search path then the **rsh** can be omitted.

OPTIONS

–l *username*
: use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username, This remote name must be equivalent (in the sense of *rlogin*(1C)) to the originating account; no provision is made for specifying a password with a command.

–n
: redirect the input of *rsh* to */dev/null.* You need this option if you are using *csh*(1) and put a *rsh*(1C) in the background without redirecting its input away from the terminal because it will block even if no reads are posted by the remote command.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1C)

BUGS

You cannot run an interactive command (like *vi*(1)); use *rlogin*(1C).

Stop signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [–a] [–l] [–t] [–u]

DESCRIPTION

Ruptime gives a status line like *uptime* for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name, but this order can be changed by specifying one of the options listed below.

OPTIONS

–a count even those users who have been idle for an hour or more.

–l sort the display by load average.

–t sort the display by up time.

–u
sort the display by number of users.

FILES

/usr/spool/rwho/whod.* data files

SEE ALSO

rwho(1C)

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [**–a**]

DESCRIPTION

The *rwho* command produces output similar to *who*(1), but for all machines on your network. If no report has been received from a machine for 5 minutes, *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, the user is omitted from the output of *rwho* unless the **–a** flag is given.

OPTIONS

–a report all users whether or not they have typed to the system in the past hour.

FILES

/usr/spool/rwho/whod.* information about other machines

SEE ALSO

ruptime(1C), rwhod(8C)

BUGS

Does not work through gateways.

This is unwieldy when the number of machines on the local net is large.

NAME

sact – print current SCCS file editing activity

SYNOPSIS

/usr/sccs/sact file ...

DESCRIPTION

Sact informs the user of any SCCS files which have had one or more **get –e** commands applied to them, that is, there are files out for editing, and deltas are pending. If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (that is, executed a *get* for editing).
Field 4	contains the date that **get –e** was executed.
Field 5	contains the time that **get –e** was executed.

SEE ALSO

sccs(1), delta(1), get(1), unget(1).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sccs – front end for the SCCS subsystem

SYNOPSIS

sccs [**–r**] [**–d***prefixpath*] [**–p***finalpath*] *command* [SCCS-flags ...] [file ...]

DESCRIPTION

The *sccs* command is a front end to the utility programs of the Source Code Control System (SCCS), which helps them mesh more cleanly with the rest of UNIX. *Sccs* runs the *command* with the specified SCCS-flags and *files*.

Note that *sccs* normally prefixes each *file*, or the last component of each *file*, with the string *SCCS/s.*, because you normally keep your SCCS database files in a directory called SCCS, and each database file starts with an *s.* prefix.

Sccs program options must appear before the *command* argument. Flags to be passed to the actual SCCS utility program must appear after the *command* argument. These flags are specific to the *command* being used, and are discussed in the manual page for that *command*.

Sccs also includes the capability to run 'set user id' to another user to provide additional protection. Certain commands (such as *admin*) cannot be run 'set user id' by all users, since this would allow anyone to change the authorizations. Such commands are always run as the real user.

OPTIONS

–r Runs *sccs* as the real user rather than as whatever effective user *sccs* is 'set user id' to.

–d*prefixpath*

Defines the prefix portion of the pathname for the SCCS database files. The default prefix portion of the pathname is the current directory. *Prefixpath* is prefixed to the entire pathname. For example:

tutorial% **sccs –d/usr/include get sys/inode.h**

converts to:

get /usr/include/sys/SCCS/s.inode.h

The intent here is to create aliases such as:

alias sysscc sccs –d/usr/src

which will be used as:

tutorial% **sysscc get cmd/who.c**

–p*finalpath*

Defines the name of a lower directory in which the SCCS files will be found; SCCS is the default. *Finalpath* is appended before the final component of the pathname. For example:

tutorial% **sccs –pprivate get usr/include/stdio.h**

converts to:

get usr/include/private/s.stdio.h

ADDITIONAL SCCS COMMANDS

Several 'pseudo-commands' are available in addition to the usual SCCS commands. These are:

admin The *admin* command is similar to that of the raw SCCS *admin* command. When creating new *s.* files, the *create* command (described just below) does more of the startup work for you and should be used in preference to *admin*.

create *Create* is used when creating new *s.* files. Given a C source language file called *obscure.c, Create* does the following actions: (1) creates the *s.* file called *s.obscure.c* in the SCCS directory; (2) renames the original source file to *,obscure.c;* (3) does an **sccs get** on *obscure.c*.

edit Get a file for editing.

delget Perform a *delta* on the named files and then *get* new versions. The new versions have id keywords expanded, and so cannot be edited.

deledit Same as *delget*, but produces new versions suitable for editing. *Deledit* is useful for making a 'checkpoint' of your current editing phase.

fix Removes the named delta, but leaves you with a copy of the delta with the changes that were in it. *Fix* must be followed by a **-r** flag. *Fix* is useful for fixing small compiler bugs, etc. Since *fix* doesn't leave audit trails, use it carefully.

clean Removes everything from the current directory that can be recreated from SCCS files. *Clean* checks for and does not remove any files being edited. If *Clean* **-b** is used, branches are **not** checked to see if they are currently being edited. Note that **-b** is dangerous if you are keeping the branches in the same directory.

unedit 'Undoes' the last *edit* or *get* **-e** and returns a file to its previous condition. If you *unedit* a file being edited, all changes made since the beginning of the editing session are lost.

info Displays a list of all files being edited. If the **-b** flag is given, branches (that is, SID's with two or fewer components) are ignored. If the **-u** flag is given (with an optional argument), only files being edited by you (or the named user) are listed.

check Checks for files currently being edited, like *info*, but returns an exit code rather than a listing: nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. *Check* may thus be included in an 'install' entry in a makefile, to ensure that everything is included in an SCCS file before a version is installed.

tell Displays a list of files being edited on the standard output. Filenames are separated by newlines. Takes the **-b** and **-u** flags like *info* and *check*.

diffs Compares (in *diff*-like format) the current version of the program you have out for editing and the versions in SCCS format.

EXAMPLES

To put a file called *myprogram.c* into SCCS format for the first time, assuming also that there is no SCCS directory already existing:

```
tutorial% mkdir SCCS
tutorial% sccs create myprogram.c

myprogram.c:
1.1
14 lines
   after you have verified that everything is all right
   you remove the version of the file that starts with a comma:
tutorial% rm ,myprogram.c
tutorial%
```

To get a copy of *myprogram.c* for editing, edit that file, then place it back in the SCCS database:

```
tutorial% sccs edit myprogram.c
1.1
new delta 1.2
14 lines
tutorial% vi myprogram.c
   your editing session
tutorial% sccs delget myprogram.c
comments? Added abusive responses for compatibility with rest of system
1.2
7 inserted
7 deleted
7 unchanged
```

```
1.2
14 lines
tutorial%
```

To *get* a file from another directory:

```
tutorial% sccs -p/usr/src/sccs/ get cc.c
```

or:

```
tutorial% sccs get /usr/src/sccs/cc.c
```

To make a delta of a large number of files in the current directory:

```
tutorial% sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
tutorial% sccs info -b
```

To *delta* everything that you are editing:

```
tutorial% sccs delta `sccs tell -u`
```

In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
        sccs get $(REL) $@
```

REGULAR SCCS COMMANDS

The 'regular' SCCS commands are described very briefly below. It is unlikely that you ever need to use these commands because the user interface is so complicated, and the *sccs* front end command does 99.9% of the interesting tasks for you.

admin Creates new SCCS files and changes parameters of exitsing SCCS files. You can use **sccs create** to create new SCCS files, or use **sccs admin** to do other things.

cdc Changes the commentary material in an SCCS delta.

comb Combines SCCS deltas and reconstructs the SCCS files.

delta Permanently introduces changes that were made to a file previously retrieved via **sccs get**. You can use **sccs delget** as the more useful version of this command since **sccs delget** does all of the useful work and more to boot.

get Extracts a file from the SCCS database, either for compilation, or for editing when the **-e** option is used. Use **sccs get** if you really need it, but **sccs delget** will normally have done this job for you. Use **sccs edit** instead of **get** with the **-e** option.

help Supposed to help you interpret SCCS error messages, but usually just parrots the messaage and is generally not considered very helpful.

prs Displays information about what is happening in an SCCS file.

rmdel Removes a delta from an SCCS file.

sact Displays information about editing activity in an SCCS file. The **sccs info** command is a lot more useful for the real life user.

sccsdiff Compares two versions of an SCCS file and generates the differences between the two versions. The **sccs delget** command does all this work as part of its normal process.

unget Undoes the work of a previous **get -e** command or a **sccs edit** command. Use the **sccs unedit** command as a useful alternative.

val Determines if a given SCCS file meets specified criteria. If you use the *sccs* command, you shouldn't need to use *val*, because its user interface is unbelievable.

what Displays SCCS identification information.

SEE ALSO

admin(1), comb(1), cdc(1), delta(1), get(1), help(1), prs(1), rmdel(1), sact(1), sccsdiff(1), sccsfile(5), unget(1), val(1), what(1)

***Source Code Control System* in *Programming Tools for the Sun Workstation*.**

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

/usr/sccs/sccsdiff **–r** *SID1* **–r** *SID2* [**–p**] [**–s** *n*] files

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but options apply to all files.

OPTIONS

–r*SID?* *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *diff*(1) in the order given.

–p pipe output for each file through *pr*(1).

–s*n* *n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

sccs(1), bdiff(1), get(1), help(1), pr(1).
Source Code Control System in *Programming Tools for the Sun Workstation*.

DIAGNOSTICS

"*file*: No differences" If the two versions are the same.
Use *help*(1) for explanations.

NAME

script – make typescript of terminal session

SYNOPSIS

script [–a] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the **–a** option is given. It can be sent to the line printer later with *lpr*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

OPTIONS

Append the script to the specified file instead of writing over it.

BUGS

Script places **everything** in the log file. This is not what the naive user expects.

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** *script*] [**–f** *sfile*] [file] ...

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands.

OPTIONS

–e Is a list of edit commands for *sed.* If there is just one **–e** option and no **–f**'s, the **–e** flag **–e** may be omitted.

–f Take the script from *sfile*

–n suppress the default output.

SED SCRIPTS

sed scripts consist of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a 'D' command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

An *address* is either a decimal number that counts input lines cumulatively across files, a '$' that addresses the last line of input, or a context address, '/regular expression/', in the style of *ed*(1) modified thus:

The escape sequence '\n' matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. If the second address is a number less than or equal to the line number first selected, only one line is selected. Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function '!' (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

An argument denoted *text* consists of one or more lines, all but the last of which end with '\' to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an 's' command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An argument denoted *rfile* or *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a\
text

Append: Place *text* on the output before reading the next input line.

(2) b *label*

Branch to the ':' command bearing the *label.* Branch to the end of the script if *label* is empty.

(2)c\
text

Change: Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2)d Delete the pattern space. Start the next cycle.

(2)D Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2)g Replace the contents of the pattern space by the contents of the hold space.

(2)G Append the contents of the hold space to the pattern space.

(2)h Replace the contents of the hold space by the contents of the pattern space.

(2)H Append the contents of the pattern space to the hold space.

(1)i\
text

Insert: Place *text* on the standard output.

(2)n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2)N Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2)p Print: Copy the pattern space to the standard output.

(2)P Copy the initial segment of the pattern space through the first newline to the standard output.

(1)q Quit: Branch to the end of the script. Do not start a new cycle.

(2)r *rfile*

Read the contents of *rfile*. Place them on the output before reading the next input line.

(2)s/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of '/'. For a fuller description see *ed*(1). *Flags* is zero or more of

g Global: Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write: Append the pattern space to *wfile* if a replacement was made.

(2)t *label*

Test: Branch to the ':' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a 't'. If *label* is empty, branch to the end of the script.

(2)w *wfile*

Write: Append the pattern space to *wfile*.

(2)x Exchange the contents of the pattern and hold spaces.

(2)y/*string1*/*string2*/

Transform: Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*

Don't: Apply the *function* (or group, if *function* is '{') only to lines *not* selected by the address(es).

(0): *label*
This command does nothing; it bears a *label* for 'b' and 't' commands to branch to.

(1)= Place the current line number on the standard output as a line.

(2){ Execute the following commands through a matching '}' only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO

ed(1), grep(1), awk(1), lex(1)
Using sed, the Stream Text Editor in *Editing and Text Processing on the Sun Workstation.*

NAME

sh, for, case, if, while, :, ., break, continue, cd, eval, exec, exit, export, login, newgrp, read, readonly, set, shift, times, trap, umask, wait – command language

SYNOPSIS

sh [**–celknrstuvx**] [**arg**] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See **Invocation** for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non blank *words* separated by blanks (a blank is a **tab** or a **space**). The first word specifies the name of the command to be executed. Except as specified below the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *execve*(2)). The *value* of a simple-command is its exit status if it terminates normally or 200+ *status* if it terminates abnormally (see *sigvec*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more *pipelines* separated by **;**, **&**, **&&** or **||** and optionally terminated by **;** or **&**. **;** and **&** have equal precedence which is lower than that of **&&** and **||**; **&&** and **||** also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol **&&** causes the *list* following to be executed only if the preceding *pipeline* returns a zero value. The symbol **||** causes the *list* following to be executed only if the preceding *pipeline* returns a non zero value. Newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. The value returned by a command is that of the last simple-command executed in the command. Note that many keywords such as **done** are only recognized when they are the first keyword on a line.

for *name* [**in** *word* ...]
do
list
done

Each time a **for** command is executed *name* is set to the next word in the **in** *word* list. If **in** *word*... is omitted, **in $@** is assumed. Execution ends when there are no more words in the list.

case *word* **in**
[*pattern* [| *pattern*] ...) *list* **;;**]
[*pattern* [| *pattern*] ...) *list* **;;**]
...
esac

A **case** command executes the *list* associated with the first pattern that matches *word*. The form of the patterns is the same as that used for file name generation.

if *list*
then *list*
[**elif** *list*
then *list*] ...
[**else**
list]
fi

The *list* following **if** is executed and if it returns zero the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero the *list* following **then** is executed. Failing that the **else** *list* is executed.

while *list*
[**do**
list]
done

A **while** command repeatedly executes the **while** *list* and if its value is zero executes the **do** *list;* otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list.* **until** may be used in place of **while** to negate the loop termination test.

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

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

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

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

Command substitution.
The standard output from a command enclosed in a pair of back quotes (``) may be used as part or all of a word; trailing newlines are removed.

Parameter substitution.
The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set**. Variables may be set by writing

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

${*parameter***}**
A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters * @ # ? - $!. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is a digit, it is a positional parameter. If *parameter* is * or @ then all the positional parameters, starting with $1, are substituted separated by spaces. $0 is set from argument zero when the shell is invoked.

${*parameter***-***word***}**
If *parameter* is set, substitute its value; otherwise substitute *word.*

${*parameter***=***word***}**
If *parameter* is not set, set it to *word;* the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***?***word***}**
If *parameter* is set, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

${*parameter***+***word***}**
If *parameter* is set, substitute *word;* otherwise substitute nothing.

In the above *word* is not evaluated unless it is to be used as the substituted string. (So that, for example, echo ${d-`pwd`} will only execute *pwd* if *d* is unset.)

The following *parameters* are automatically set by the shell.

#	The number of positional parameters in decimal.
-	Options supplied to the shell on invocation or by **set**.
?	The value returned by the last executed command in decimal.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following *parameters* are used but not set by the shell.

HOME The default argument (home directory) for the **cd** command.
PATH The search path for commands (see **execution**).
MAIL If this variable is set to the name of a mail file, the shell informs the user of the arrival of mail in the specified file.
PS1 Primary prompt string, by default '$ '.
PS2 Secondary prompt string, by default '> '.
IFS Internal field separators, normally **space**, **tab**, and **newline**.

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

File name generation.
Following substitution, each command word is scanned for the characters *, ? and [. If one of these characters appears, the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, and the character /, must be matched explicitly.

* Matches any string, including the null string.
? Matches any single character.
[...] Matches any one of the characters enclosed. A pair of characters separated by – matches any character lexically between the pair.

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

; & ( ) | < > newline space tab

A character may be *quoted* by preceding it with a backslash (\) character. \newline is ignored. All characters enclosed between a pair of quote marks (''), except a single quote, are quoted. Inside double quotes ("") parameter and command substitution occurs and \ quotes the characters backslash (\), apostrophe ('), double quote ("), and dollar sign ($).

"$*" is equivalent to "$1 $2 ..." whereas
"$@" is equivalent to "$1" "$2" ...

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

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

< *word* Use file *word* as standard input (file descriptor 0).

> *word* Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise it is truncated to zero length.

>> *word*
Use file *word* as standard output. If the file exists, output is appended (by seeking to the end); otherwise the file is created.

<< word

The shell input is read up to a line the same as *word*, or end of file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\newline** is ignored, and **** is used to quote the characters backslash (****), dollar sign (**$**), apostrophe (**'**), and the first character of *word*.

< & digit

The standard input is duplicated from file descriptor *digit;* see *dup*(2). Similarly for the standard output using **>**.

< &- The standard input is closed. Similarly for the standard output using **>**.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example,

```
... 2>&1
```

creates file descriptor 2 to be a duplicate of file descriptor 1.

If a command is followed by **&** then the default standard input for the command is the empty file (/dev/null). Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input output specifications.

Environment.

The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see *execve*(2) and *environ*(5). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a *parameter* for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these *parameters* or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's *parameter* to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to *parameters*. Thus these two lines are equivalent

```
TERM=450 cmd args
(export TERM; TERM=450; cmd args)
```

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following prints 'a=b c' and 'c':

```
echo a=b c
set -k
echo a=b c
```

Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent. (But see also **trap**.)

Execution.

Each time a command is executed the above substitutions are carried out. Except for the 'special commands' listed below a new process is created and an attempt is made to execute the command via an *execve*(2).

The shell parameter **$PATH** defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (**:**). The default path is **:/bin:/usr/bin**. If the command name contains a **/**, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an *a.out* file, it

is assumed to be a file containing shell commands. A subshell (that is, a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands.
The following commands are executed in the shell process and except where specified no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. *file* Read and execute commands from *file* and return. The search path $PATH is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter $HOME is the default *arg*.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input output arguments may appear and if no other arguments are given cause the shell input output to be modified.

exit [*n*]
Causes a non interactive shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. (An end of file will also exit from the shell.)

export [*name* ...]
The given names are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of exportable names is printed.

login [*arg* ...]
Equivalent to 'exec login arg ...'.

newgrp [*arg* ...]
Equivalent to 'exec newgrp arg ...'.

read *name* ...
One line is read from the standard input; successive words of the input are assigned to the variables *name* in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

readonly [*name* ...]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given, a list of all readonly names is printed.

set [**–eknptuvx** [*arg* ...]]

–e If non interactive, exit immediately if a command fails.

–k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

–n Read commands but do not execute them.

–t Exit after reading and executing one command.

–u Treat unset variables as an error when substituting.

–v Print shell input lines as they are read.

–x Print commands and their arguments as they are executed.

– Turn off the **–x** and **–v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$-**.

Remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, etc. If no arguments are given, the values of all names are printed.

shift The positional parameters from **$2**... are renamed **$1**...

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
Arg is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by invoked commands. If *n* is 0, the command *arg* is executed on exit from the shell, otherwise upon receipt of signal *n* as numbered in *sigvec*(2). *Trap* with no arguments prints a list of commands associated with each signal number.

umask [*nnn*]
The user file creation mask is set to the octal value *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is that of the process waited for.

Invocation.
If the first character of argument zero is –, commands are read from **$HOME/.profile**, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

–c *string* If the –c flag is present, commands are read from *string*.

–s If the –s flag is present or if no arguments remain then commands are read from the standard input. Shell output is written to file descriptor 2.

–i If the –i flag is present or if the shell input and output are attached to a terminal (as told by *gtty*) then this shell is *interactive*. In this case the terminate signal SIGTERM (see *sigvec*(2)) is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptable). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command.

FILES

$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

Using the Bourne Shell in the *Beginner's Guide to the Sun Workstation*
csh(1), test(1), execve(2).

DIAGNOSTICS

Errors detected by the shell, such as syntax errors, cause the shell to return a non zero exit status. If the shell is being used non interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

BUGS

If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document. A garbage file /tmp/sh* is created, and the shell complains about not being able to find the file by another name.

NAME

size – size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each object-file argument. If no file is specified, *a.out* is used.

SEE ALSO

a.out(5)

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

setitimer(2), sleep(3)

BUGS

Time must be less than 2,147,483,647 seconds.

NAME

soelim – eliminate .so's from nroff input

SYNOPSIS

soelim [file ...]

DESCRIPTION

Soelim reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

.so somefile

when they appear at the beginning of input lines. This is useful since programs such as *tbl* do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using ' '' instead of '.', that is,

'so /usr/lib/tmac.s

EXAMPLE

A sample usage of *soelim* would be

soelim exum?.n | tbl | nroff –ms | col | lpr

SEE ALSO

colcrt(1), more(1)

BUGS

The format of the source commands must involve no strangeness – exactly one space must precede and no spaces follow the file name.

NAME

sort – sort or merge files

SYNOPSIS

sort [**–mubdfinrt***x*] [**+***pos1* [**–***pos2*]] ... [**–o** name] [**–T** directory] [file] ...

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name '–' means the standard input. If no input *file*'s are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence.

The notation **+***pos1* **–***pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2*. A missing *.n* means .0; a missing **–***pos2* means the end of the line. Under the **–t***x* option, fields are strings separated by *x*; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

OPTIONS

The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d 'Dictionary' order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b.**

r Reverse the sense of comparisons.

t*x* 'Tab character' separating fields is *x*.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

o *name*

name is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

T *directory*

directory argument is the name of a directory in which temporary files should be made.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort -u + 0f + 0 list
```

Print the password file (*passwd*(5)) sorted by user id number (the 3rd colon-separated field).

```
sort -t: + 2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um + 0 -1 dates
```

FILES

/usr/tmp/stm*, /tmp/* first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option -c.

BUGS

Very long lines are silently truncated.

NAME

sortbib – sort bibliographic database

SYNOPSIS

sortbib [–sKEYS] database ...

DESCRIPTION

Sortbib sorts files of records containing *refer* key-letters by user-specified keys. Records may be separated by blank lines, or by .[and .] delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so *sortbib* may not be used in a pipeline to read standard input.

By default, *sortbib* alphabetizes by the first %A and the %D fields, which contain the senior author and date. The –s option is used to specify new KEYS. For instance, –sATD will sort by author, title, and date, while –sA+D will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

Sortbib sorts on the last word on the %A line, which is assumed to be the author's last name. A word in the final position, such as "jr." or "ed.", will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the *nroff* convention "\0" in place of a blank. A %Q field is considered to be the same as %A, except sorting begins with the first, not the last, word. *Sortbib* sorts on the last word of the %D line, usually the year. It also ignores leading articles (like "A" or "The") when sorting by titles in the %T or %J fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, *sortbib* places that record before other records containing that field.

SEE ALSO

refer(1), addbib(1), roffbib(1), indxbib(1), lookbib(1)

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell, spellin, spellout – find spelling errors

SYNOPSIS

spell [**–v**] [**–b**] [**–d** hlist] [**–s** hstop] [**–h** spellhist] [file] ...

spellin [list]

spellout [**–d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (for example, thier=thy–y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell*. Both expect a list of words, one per line, from the standard input. *Spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option –d) the hash list.

OPTIONS

–v Print all words not literally in the spelling list, as well as plausible derivations from spelling list words.

–b Check British spelling. Besides preferring *centre, colour, speciality, travelled*, and so on, the –b option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

–x print every plausible stem with '=' for each word.

FILES

/usr/dict/hlist[ab]	hashed spelling lists, American & British
/usr/dict/hstop	hashed stop list
/usr/dict/spellhist	history file
/usr/dict/words	list of words
/usr/lib/spell	

SEE ALSO

deroff(1), sort(1), tee(1), sed(1)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.

British spelling was done by an American.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [**–a**] [**–k**] [**–n**] [**–p**] [**–x**]

DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

OPTIONS

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

–k The constant k used in the boundary value computation

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument. By default $k = 0$.

–n Space output points so that approximately n intervals occur between the lower and upper x limits. (Default $n = 100$.)

–p Make output periodic, that is, match derivatives at ends. First and last input values should normally agree.

–x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1G)

DIAGNOSTICS

When data is not strictly monotonic in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

NAME

split – split a file into pieces

SYNOPSIS

split [**–n**] [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000) onto a set of output files (as many files as necessary). The name of the first output file is *name* with **aa** appended, the second file is *name***ab**, and so on lexicographically.

If no *name* is given, *x* is used as default (output files will be called *x***aa**, *x***ab**, etc.).

If no input file is given, or if – is given in its stead, then the standard input file is used.

OPTIONS

–n Number of lines in each piece.

NAME

strings – find printable strings in an object, or other binary, file

SYNOPSIS

strings [–] [**–o**] [*–number*] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null.

Strings is useful for identifying random object files and many other things.

OPTIONS

– Look everywhere in the file for strings. If this flag is omitted, *strings* only looks in the initialized data space of object files.

–o Precede each string by its offset in the file (in octal).

–number
Use *number* as the minimum string length rather than 4.

SEE ALSO

od(1)

BUGS

The algorithm for identifying strings is extremely primitive.

NAME

strip – remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **–s** option of *ld*.

FILES

/tmp/stm? temporary file

SEE ALSO

ld(1)

NAME

stty – set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, and directs its output to the diagnostic output. With no argument, it reports the speed of the terminal and the settings of options which are different from their defaults. With the argument "all", all normally-used option settings are reported. With the argument "everything", everything *stty* knows about is printed.

OPTIONS

Options to *stty* are selected from the following set:

even allow even parity input
–even disallow even parity input
odd allow odd parity input
–odd disallow odd parity input
raw raw mode input (no input processing (erase, kill, interrupt, ...); parity bit passed back)
–raw negate raw mode
cooked same as '–raw'
cbreak make each character available to *read*(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed
–cbreak make characters available to *read* only when newline is received
–nl allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl accept only new-line to end lines
echo echo back every character typed
–echo do not echo characters
lcase map upper case to lower case
–lcase do not map case
tandem enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
–tandem disable flow control
–tabs replace tabs by spaces when printing
tabs preserve tabs
ek set erase and kill characters to ^H (control-H) and ^U

For the following commands which take a character argument *c*, you may also specify *c* as the "u" or "undef", to set the value to be undefined. A value of "^x", a 2 character sequence, is also interpreted as a control character, with "^?" representing delete.

erase *c* set erase character to *c* (default '^?').
kill *c* set kill character to *c* (default '^U').
intr *c* set interrupt character to *c* (default '^C').
quit *c* set quit character to *c* (default '^\').
start *c* set start character to *c* (default '^Q').
stop *c* set stop character to *c* (default '^S').
eof *c* set end of file character to *c* (default '^D').
brk *c* set break character to *c* (default undefined.) This character is an extra wakeup causing character.
cr0 cr1 cr2 cr3
select style of delay for carriage return (see *ioctl*(2))
nl0 nl1 nl2 nl3
select style of delay for linefeed
tab0 tab1 tab2 tab3
select style of delay for tab

ff0 ff1 select style of delay for form feed
bs0 bs1 select style of delay for backspace

tty33 set all modes suitable for the Teletype Corporation Model 33 terminal.
tty37 set all modes suitable for the Teletype Corporation Model 37 terminal.
vt05 set all modes suitable for Digital Equipment Corp. VT05 terminal
dec set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and "newcrt".)

tn300 set all modes suitable for a General Electric TermiNet 300
ti700 set all modes suitable for Texas Instruments 700 series terminal
tek set all modes suitable for Tektronix 4014 terminal
0 hang up phone line immediately
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

The driver which supports the job control processing of *csh*(1) is fully described in *tty*(4). The options in the list below can only be selected by using the **new** option to *stty*(1).

new Use new driver (switching flushes typeahead).
crt Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)
crtbs Echo backspaces on erase characters.
prterase For printing terminal echo erased characters backwards within "\" and "/".
crterase Wipe out erased characters with "backspace-space-backspace."
-crterase Leave erased characters visible; just backspace.
crtkill Wipe out input on like kill ala **crterase.**
-crtkill Just echo line kill character and a newline on line kill.
ctlecho Echo control characters as "^*x*" (and delete as "^?".) Print two backspaces following the EOT character (default '^D').
-ctlecho Control characters echo as themselves; in cooked mode EOT (default '^D') is not echoed.

decctlq After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.

-decctlq After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)
tostop Background jobs stop if they attempt terminal output.
-tostop Output from background jobs to the terminal is allowed.
tilde Convert "~" to "`" on output (for Hazeltine terminals).
-tilde Leave poor "~" alone.
flusho Output is being discarded usually because user hit '^O' (internal state bit).
-flusho Output is not being discarded.
pendin Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).
-pendin Input is not pending.
intrup Send a signal (SIGTINT) to the terminal control process group whenever an input record (line in cooked mode, character in cbreak or raw mode) is available for reading.
-intrup Don't send input available interrupts.
mdmbuf Start/stop output on carrier transitions (not implemented).
-mdmbuf
Return error if write attempted after carrier drops.
litout Send output characters without any processing.
-litout Do normal output processing, inserting delays, etc.
nohang Don't send hangup signal if carrier drops.
-nohang Send hangup signal to control process group when carrier drops.

etxack Diablo style etx/ack handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed.

susp *c* set suspend process character to *c* (default '^Z').
dsusp *c* set delayed suspend process character to *c* (default '^Y').
rprnt *c* set reprint line character to *c* (default '^R').
flush *c* set flush output character to *c* (default '^O').
werase *c* set word erase character to *c* (default '^W').
lnext *c* set literal next character to *c* (default '^V').

SEE ALSO

ioctl(2), tset(1), tty(4)

NAME

su – substitute user id temporarily

SYNOPSIS

su [*username*] [**–c** *command*]

DESCRIPTION

Su changes your login to that of the specified *username*. *Su* asks for the password, just as if you were logging in as *username*, and, if the password is given, changes to that *username* and invokes the shell specified in the password file for that *username*, without changing the current directory. The user environment is thus unchanged except for HOME and SHELL, which are taken from the password file for the user being substituted (see *environ*(5)). The new user ID stays in force until the shell exits.

If no *username* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

OPTIONS

–c *command*

execute *command* after logging in as the new user.

SEE ALSO

sh(1), csh(1)

NAME

sum – sum and count blocks in a file

SYNOPSIS

sum file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinguishable from end of file on most devices; check the block count.

NAME

sun – is current machine a sun workstation

SYNOPSIS

```
if sun; then ...; fi                (sh)

if { sun } then                     (csh)
    ...
endif
```

DESCRIPTION

The *sun* command is, on Sun Workstations, the same as *true*(1); on VAX'es and other machines it is the same as *false*(1).

SEE ALSO

false(1), true(1), vax(1)

NAME

suntools – the Suntools window environment

SYNOPSIS

[setenv PATH ${PATH}:/usr/suntool]
suntools [**–n** | **–s** *start-up file*] [**–d** *display device*] [**–m** *mouse device*] [**–k** *keyboard device*] [**–i**] [**–f** *red green blue*] [**–b** *red green blue*] [**–F** | **–B** | **–P**] [**–p**]

GETTING STARTED

Suntools starts up the Suntools window environment and awaits the user's directions (often a menu action). If */dev* has not been initialized for the window environment, *suntools* will fail; see *Initialization* below for details. If there is a file called *.suntools* in the user's home directory, *suntools* starts the application programs (*tools*) specified in that file. The format of the *.suntools* file is described in detail below under *Start-up Processing.*

The indications that the *suntools* system is ready are:

1) the screen is painted with a solid color (usually light gray, see OPTIONS), and

2) an arrow-shaped cursor appears on the screen, tracking motions of the mouse. The mouse must be on its special pad before it can be tracked. Sometimes the cursor will not move at start-up; moving the mouse in large circles for a few seconds will bring the cursor to life.

See *Multiple/Color Displays* for information about running *suntools* on more than one display at the same time or on color displays.

OPTIONS

–n Start-up processing is bypassed by specifying this flag.

–s *start-up file*
Specifies a file from which start-up commands are read.

–d *display device*
Specifies a display device on which *suntools* is run. If this option is not specified, the default frame buffer device, */dev/fb*, is used.

–m *mouse device*
Specifies a mouse device used by *suntools* as the system printing/locator device. If this option is not specified, the default mouse device, */dev/mouse*, is used.

–k *keyboard device*
Specifies a keyboard device used by *suntools* as the system keyboard. If this option is not specified, the default keyboard device, */dev/kbd*, is used.

–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.

–f *red green blue*
Specifies the values of the *red, green and blue* components of the foreground color. For most color screens, values must be between 0 and 255 where 0 is the absence of color and 255 is the maximum intensity of that color. If this option is not specified, each component of the foreground color is 0 (black).

–b *red green blue*
Specifies the values of the *red, green and blue* components of the background color. If this option is not specified, each component of the background color is 255 (white). See the **–f** option for more details.

–F Use the foreground color for the *suntools* window color.

-B Use the background color for the *suntools* window color.

-P Use a stipple pattern for the *suntools* window color. If no **-F**, **-B** or **-P** option is given, **-P** is assumed.

-p Display the name of the window device used for the *suntools* window.

DESCRIPTION

Windows

The *suntools* window environment always has one window open which covers the whole screen; this is called the *Root Window*. A solid color is the only content of the Root Window; it serves as a *backdrop* for the *suntools* environment. Tools that run in the environment are given their own windows which lie on top of portions of the Root Window (and possibly other tools). A window obscures those portions of other windows which lie below it; thus, only the topmost window is guaranteed to be fully visible, and the Root Window color shows through only in areas which have no tool windows on them.

User Input to Windows

User input actions (typing on the keyboard, moving the mouse, or pressing buttons on the mouse) are directed to the window which lies immediately under the cursor at the time they occur. User input actions are queued for each window, with mouse motions and button pushes integrated with keystrokes. It is perfectly reasonable to type some characters to one window, then move the mouse to another window, and type or use the mouse in it before the first window has processed its input; the *type-ahead* and *mouse-ahead* are serialized and distributed correctly.

The mouse buttons are usually interpreted as follows: the left button is used for selecting or choosing objects; the middle button is used to adjust selections to include more or fewer objects; and the right button is used to invoke menus. However, the exact interpretation of user actions depends on the window receiving them. For the standard tools available with the *suntools* system, the tool's input handling is described with the rest of its behavior in its specific section below.

Menus

Many functions in *suntools* may be invoked by using the mouse and a *pop-up menu*. By convention, pressing the *menu* (right) button on the mouse and holding it down calls up the menu(s) appropriate to the window containing the cursor. A menu is painted on the screen near the current location of the mouse, and the cursor is changed to a right-pointing arrow. The menu remains on the screen as long as the menu button is held down, and the pointer tracks the mouse. When the pointer enters a menu item, that item video-inverts, but nothing else happens while the menu button is kept down. Releasing the button while the pointer is in a menu item invokes that menu item: the menu is removed from the screen, and processing of that item begins.

Multiple Menus

Multiple menus are currently available only in the *panetool*. However, the following description applies to multiple menus in general.

You may see more than one menu presented simultaneously; generally this occurs when several different classes of actions are possible in the current context. The menus are presented in a stack, with the label of each menu visible, and with the current menu on top so its items are also visible. To bring another menu to the top, thus making its items available, invoke its label as you would a menu item. That is, position the cursor in the label of the desired menu and release the menu button. Then press the menu button again, and the stack of menus is presented once more, but now with the selected menu on top.

An *accelerator* is provided for expert users to achieve the same result a little more quickly: when the cursor is positioned in the label of the desired menu, *do not release the menu button*; rather, while continuing to hold down the menu button, press and release the *select* (left) mouse button. The stack of menus is repainted with the selected menu on top, and menu operations may continue.

The Root Manager

The Root Window ignores most keyboard input; any keystrokes generated while the cursor is over an exposed portion of the Root Window are simply discarded. However, four job-control functions are provided through the Root Window's menu, which is presented when the menu button is depressed while the cursor is over an exposed portion of the Root Window.

The items in the Root Manager menu are:

New Shell — Create a new Shell Tool.
A new tool window is created on the screen, running a new copy of the shell. The window is placed on the screen on top of everything it overlaps. Its initial size can display 80 columns and 34 rows of characters.

New Graphics — Create a new Graphics Tool.
A new tool window appropriate for running graphics programs is placed on the screen on top of everything it overlaps.

Exit — Exit the *suntools* program.
All tool windows are closed, and their associated processes are killed. The user is returned to the shell which invoked *suntools*. This command requires confirmation: press the left mouse button to complete it; press the right button to cancel.

ReDisplay All — All the contents of the screen are redrawn.
This can be used to repair damage done by processes that wrote to the screen without consulting the Suntools system (see the discussion of the Shell Tool for details on how to avoid this).

The Tool Manager

Tools interpret most user input in their own fashion; for instance, many tools incorporate a terminal emulator running a shell. However, tools are also expected to provide a small set of universal functions, through the Tool Manager menu. These functions are available when the cursor is over a portion of the tool which does not impose some other interpretation on the user input. For any tool, the black stripe at the top which holds the tool's name, the border stripes of the window, and the whole of the tool's icon are such areas.

The items in this menu are:

Close (or Open) — Shrink the tool to a small image on the screen.
Its process(es) continues to run. Closed windows are by default placed in the lower left of the screen, filling in rows to the right, and then bottom-to-top, in the order they are created. A closed window may be moved just like an open one; if it is reopened and closed again, it returns to its last closed position.

When the Tool Manager Menu is brought up over a closed window, this item reads Open, and serves to reverse the process. When reopened, windows return to the position from which they were closed.

Move — Change the location of a window on the screen, without affecting its size or contents.
When invoked, Move instructs the user to grasp the window by depressing and holding down either the left or middle mouse buttons, or to cancel the operation by pressing the right button; depressing any mouse button removes these

instructions from the screen. When either the left or middle button is depressed, the window is outlined by a box which tracks the mouse as long as that button is held down. The position of the pointer when the button is pressed determines how the window is moved: if it is near a corner, that corner of the box is attached to the mouse cursor. If it is in the middle third of a side, then the midpoint of the side of the box is attached to the mouse cursor. This box should be positioned where the window should go and the mouse button released; the window is repainted in its new location, and its old location repaired. The level of the window (on top of or underneath others) is not affected.

Stretch — Change the size of a window on the screen.
As with Move, instructions are posted for the user until a mouse button push either starts the Stretch or cancels it. The position of the pointer when either the left or middle button is pressed determines how the window is stretched: if it is near a corner, both of the sides that form that corner are adjusted; the opposite corner remains fixed. If it is in the middle third of a side, then only that side is adjusted; all 3 other sides remain fixed. The use of the bounding box, and behavior with respect to window level, is the same as for Move.

Expose — Bring the window to 'the top of the heap'.
The whole window becomes visible, and occludes any window it happens to overlap on the screen. Its position on the screen does not change.

Hide — Put the window on the 'bottom of the heap'.
The window is occluded by any window which overlaps it. Its position on the screen does not change.

ReDisplay — Redraw the contents of the window.

Quit — Notify the tool that it should terminate gracefully.
This command requires the same type of confirmation as the Exit command in the Root Menu.

In many multi-subwindow tools, the boundary between two subwindows may be adjusted up or down without changing the overall size of the tool. To do this, depress the middle mouse button over the boundary; a bounding box will be drawn for the subwindow above or to the left of the selected boundary. Now adjust the size of that subwindow, exactly as with the Stretch operation. When the button is released, that subwindow is adjusted to the size indicated, and the remaining area of the tool is allotted to the other subwindows.

Tool Management Accelerators

Accelerators are provided for some of the Tool Manager functions, allowing experienced users to invoke these functions more quickly: these functions can be invoked with a simple button push in the tool window.

The accelerators for the various functions are:

Open — A closed (iconic) tool may be opened by clicking the select (left) mouse button while the cursor is over the icon.

Move — The *Move* operation may be invoked by depressing the middle mouse button while the cursor is in the tool window's name stripe or outer boundary; a bounding box is displayed and tracks the mouse as long as the middle button is held down; the window is placed in the position indicated by that box when the button is released.

Expose — Clicking the left mouse button at an open tool window exposes that window.

Help — Typing a question mark to a tool window will cause a help message to be displayed on the screen, explaining the material covered here.

Terminal Emulators and Selections

Several tools provide a terminal emulator in some part of their window, running a shell. Keystrokes typed to this *subwindow* are passed through to the program running in it, rather than being handled by the tool. (This means these tools provide Tool Manager functions only when the cursor is in their name stripe or borders.) Similarly, output from the programs is displayed in the window.

Terminal subwindows support a facility called a *selection,* which provides for limited inter-tool communication and mouse-oriented text manipulation. A *selection* is a span of characters which you may wish to manipulate. The mouse is used to make a selection:

- Click the *select* (left) mouse button while the tip of the cursor is over a character, and that character is selected. Any previously selected characters are de-selected. A box is drawn around selected characters to indicate their status.
- Click the *adjust* (middle) mouse button to change the span of characters that are selected: all characters from the one originally selected through the one over which *adjust* is pressed are selected. Characters which used to be selected, but which do not lie in the span just described, are de-selected. The box which indicates the selection is adjusted to reflect its new contents.

Only characters which are displayed in a terminal subwindow may be selected. Characters which are in a portion of the window obscured by another window will be selected if they lie within a span of characters whose endpoints are selected.

The user feedback (the boxed characters) indicating the selection is removed if:

- the cursor moves out of the subwindow which holds the selection,
- any key is typed, or
- any new output is written to the window which holds the selection.

However, even if the feedback is removed, it is important to note that *the selection still exists* and can be operated upon until the point in time that another selection is explicitly made by the user. The fact that the feedback is removed in these circumstances reflects an incomplete selection implementation.

The selection is manipulated via the *Selection* menu provided by the terminal subwindow. There is a single menu item:

Stuff — The characters in the selection, are copied to the window as though they had been typed at the keyboard. The window in which Stuff is invoked does not have to be the same as the one in which the selection was made; thus text may be copied between windows by this facility.

The selection mechanism is not integrated with the terminal emulator. However, it is useful for building up complex command sequences and helps avoid repetitive keyboard input. Selection facilities will be expanded in future releases.

Other Aspects of Terminal Emulators

Existing tools with terminal emulator subwindows pass command line arguments to the terminal emulator. The terminal emulator then passes a command line to the program it starts. No arguments indicate that the program described by the user's SHELL environment value is run by the terminal emulator. If this program is not available then */bin/sh* is run. If there are arguments, the program named by the first one is run. However, if the first argument is **–c** a shell is run which in turn runs the command line named by the argument following the **–c**. See *Start-up Processing* below for an example.

The Shell Tool

The Shell Tool is one of the standard tools provided with the *suntools* environment. It consists of a terminal subwindow, in which a shell is started. Because keystrokes are passed to the shell, or the programs run from it, and the mouse is used to support selections, the Tool Manager is not accessible from the terminal subwindow of the Shell Tool. Tool Manager functions may be accessed in the tool's name stripe and borders.

Programs that are run in the Shell Tool are generally for character-oriented terminals. Graphics Programs (see below) will run in the Shell Tool, but may exhibit minor glitches. Graphics Programs should be run in the Graphics Tool (see below).

The size of the Shell Tool's virtual terminal (in lines and columns) is adjusted to reflect the dimensions of the window as they change. Thus, programs like *emacs*, *more*, and *vi*, which inquire the characteristics of the terminal they run on, adjust to differently-sized windows automatically. Since they only inquire as they start up, however, they do not reflect changes to window dimensions that occur after they are started; such changes may result in a confusing display for that program.

One important feature of a Shell Tool is that it may be used as a substitute for the standard console. Unless instructed otherwise, the system displays messages for the console by writing directly on the bitmap display, bypassing all of the support for multiple windows sharing the bitmap; thus, console messages damage the window images. It is possible to redirect console messages into a terminal subwindow by starting a Shell Tool with a first argument of CONSOLE, which causes all messages destined for the console to appear inside this shell's window. An example appears under *Start-up Processing* below.

The Graphics Tool

The Graphics Tool is a second tool provided with *suntools*. It has two subwindows: a terminal subwindow in which a shell is started, just as for the Shell Tool, and a graphics area which may be drawn on by programs invoked from that shell. The boundary between these two subwindows may be moved as described in *The Tool Manager* above. The graphics area normally acts as part of the tool window for user input; that is, it provides Tool Manager functions to mouse and keyboard events. Some graphics programs run in this window may take over inputs directed to it; SunCore is an example, since it uses the mouse and keyboard for its own input. While this is true, Tool Manager functions are accessible only in the namestripe and borders, as with the Shell Tool. When the program which has taken the input terminates, the window is restored to its original status.

Demos which draw pictures should be run in the Graphics Tool; the four distributed with the current release of *suntools* are:

/usr/suntool/bouncedemo
: Displays a bouncing square in the graphics window. A decimal number on the command line indicates how many repetitions of the bounce sequence should be done.

/usr/suntool/spheresdemo
: Laboriously computes a random collection of shaded spheres. A –n followed by a decimal number on the command line indicates how many spheres should be drawn. Colored spheres are drawn on color displays.

/usr/suntool/jumpdemo
: Simulates the famous *Star Wars* jump to light-speed sequence using vector drawing. A –n followed by a decimal number on the command line indicates how many stars should be used in the star field, up to 500. Colored stars are drawn on color displays. A –c on the command line directs the program to rotate the color map, thus producing a sparkling affect.

/usr/suntool/framedemo

Displays a series of frames, each of which contains a 256 by 256 image formed of pixels one deep (that is, the image is a square monochrome bitmap, with 256 bits on a side). The frames must be in the files *frame.1* through *frame.n* on the current working directory, and are displayed in numerical order. A set of sample frames is available on the directory */usr/suntool/globeframes*. A –n followed by a decimal number on the command line indicates how many times to cycle through the frames. If the cursor is moved into the image window, typing certain characters affects the rate at which the frames are displayed: the initial rate is one frame per second; each "S" causes an additional one second delay between frames; each "F" removes one second from the inter-frame delay; each "s" adds 1/20th of a second; each "f" removes 1/20th of a second.

For all these demos the following is true: if no –n flag appears on the command line then the program runs continuously until interrupted by the user. A –r flag on the command line turns the window into a *retained* window which allows the image to reappear when uncovered instead of restarting the demo. These demos can all be invoked from the workstation console, i.e., outside of the *suntools* window environment. Also, a –d flag followed by a *display device* name can direct the demo to run on a display other than the console, "bouncedemo -d /dev/cgone0".

The Icon Tool

The Icon Tool is a simple bitmap editor, useful for constructing small images such as cursors and icons. It has four subwindows:

- At the top, a one-line 'text window' displays error and warning messages.
- Below that on the left, a small 'proof window' for checking the image as it is drawn,
- To the right of the 'proof window', a 'control panel' (an options subwindow, as described in chapter 7 of the *Programmer's Reference Manual for SunWindows,* and
- Below those two, filling most of the area of the tool, a large canvas on which to draw the image.

Painting

The canvas displays magnified pixels in a square either 16 or 64 on a side. The larger magnification (fewer pixels) is appropriate for cursors, the smaller, for icons. The left mouse button is used to darken (paint) pixels on the canvas, the middle button to clear (erase) them. A button may be held down and the cursor 'wiped' through several pixels. The state of the mouse buttons is lost if the cursor leaves the window; the button must be pressed after the cursor is in the window, and it must be released and pressed again if the cursor strays out of the window's boundary.

The right (menu) button has no special significance to the icontool, so it invokes the standard tool-manager menu.

Program Control

The control panel presents a number of items; the interesting ones are either parameters or commands. Each is identified by a label displayed in **bold face**. Commands are like function keys that can be 'pushed' with the mouse — clicking a mouse button over one of them causes some action to take place. They are displayed surrounded by parentheses. Parameters are either choices or text items. Choices are used to select one of a set of mutually exclusive possibilities; the set of possibilities is displayed inside braces, and the current value is displayed in reverse video. Text items have a label ending with a colon, followed by the string which is the value of that item. A very long value may not be completely displayed; characters out of sight at the end are nonetheless part of the value. The text item which is currently accepting keystrokes has a

box around its label. Since the icontool has only one text item, its label is always boxed.

In general, either the left or middle button may be used to manipulate an item in the control panel. Items give feedback (usually by inverting) when the cursor is over them with a button down to show what will happen if the button is released at that point. But they do not take any action until the button is released. You can slide the cursor around over many items until you have the one you want. As long as you don't release the button, you can cancel any action by simply moving the cursor into empty space before releasing the button.

As with the canvas window, the state of the mouse buttons is lost when the cursor crosses a window boundary: the button must be pressed after the cursor is in the options subwindow.

The particular parameters and commands are discussed in the order they appear:

Draw a: { Cursor Icon }
: Controls the size of the image the tool is working on. A cursor is 16 pixels square; and icon is 64 square. When the image type is changed, the old image is preserved, and restored if the type is later changed back. This parameter will also be set by the Load File command (see below), according to the size of the object loaded.

Left Paints, Middle Erases
: This is a simple label which does not react to user actions.

(Quit) Requests confirmation from the user, and if it is given, terminates the tool's processing. Confirm the termination by clicking the left mouse button; if you decide you don't want to quit yet, click the right or middle button.

(Load)

(Store) Allow the tool to deal with images stored as files on disk. They each use the text in the "File" parameter as the name of a file to be processed. The format of the file is an initialized C-language array of shorts or ints, with the initializers expressed in hex. (Actually, the Load routine only cares about finding the correct number of hex numbers separated by commas and surrounded by braces.)

Load reads the named file, determines its size by how many numbers it reads, sets the "Draw a" mode accordingly, and then either merges the current image of that size with bits it read, or replaces it, according to the "Load / Fill should: { Replace Merge }" parameter described below.

"Store" writes the current image into a file with the indicated name. If such a file already exists, "Store" asks for confirmation before over-writing it. Either routine complains about other anomalous conditions, such as reading a non-existent file, or violating file permissions. Write to a directory with write permissions . . . not */usr/suntools*.

File: A text parameter used by "Load" and "Store" commands as described above. Any (printable) characters typed at the keyboard while the cursor is in the option subwindow are appended to this parameter. The user's erase and kill characters may be used to delete the last character and the whole text, respectively. There is no need to enter an "insert" command or to finish the name with a Carriage Return, Escape, or other terminator; "Load" and "Store" always use the text currently available. (Non-printing characters such as CR or TAB don't do any harm either; they are simply discarded.)

(Fill) Causes a constant pattern (selected by the following choice) to be written into the current image, either replacing or adding to its current contents.

with { White 25% Root Gray 50% 75% Black }
: Governs the source for "Fill". The second through fifth values are grays of various shades and patterns; Root Gray is the pattern used for Suntools' Root Window. Try filling a cursor canvas to see what they're like.

Invert Causes all the pixels in the current image to be inverted.

Load / Fill should { Replace Merge }
: Governs the operation used by "Load" and "Fill": if it is Replace, the new image replaces the old one; if it is Merge, they are ORed together.

Cursor Op
: Only appears when you are drawing a cursor. In this case, the cursor that tracks the mouse while it is in the Proof window is the one you have drawn, and this choice controls the raster operation used to display it there.

Proof background { White 25% Root Gray 50% 75% Black }
: Controls the background pattern in the Proof window against which a cursor or icon will be displayed. The choices are the same as for "Fill's" source.

Feedback
: In icon mode, the Proof window in the upper left corner presents the current image at its true size on a background chosen by the Proof background option. In cursor mode, the whole proof window is filled with the background pattern, and the cursor in the proof window reflects the current image.

The Clock Tool

The Clock Tool is a simple tool to display the current time. It must be started from a shell, or an entry made for it in your start-up file, since it does not appear in the Root Manager menu. Type **clocktool &** to any shell running in a window; the ampersand allows the shell to process other commands while the clock is running. While it is open, the Clock Tool prints the date and time in its window; when it is closed, its icon is a clock face which keeps time. The Clock Tool's window is not a terminal subwindow; it provides the Tool Manager menu if the menu button is depressed over it. However, it listens for keyboard input, toggling its state on two letters (case does not matter):

s toggles the display of seconds. 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 toggles the 'test' mode. The clock starts with testing off. With testing on, the clock ignores the real time, and instead runs in a loop continuously incrementing the time by one minute and displaying it.

These states may only be modified while the clock is open; when it is iconic, it responds with the standard Tool Manager functions to mouse and keyboard input.

The Pane Tool

The Pane Tool is a sample tool to demonstrate a random collections of features not used by other tools:

- There are four subwindows that tile the tool surface. The boundaries between the subwindow can be moved as described above.
- Keyboard input directed at the upper right subwindow is redirected to the upper left subwindow.
- Keyboard input directed at the upper left subwindow demonstrates raw unencoded up/down keyboard codes.
- The lower left subwindow cycles through alternating menu stacks each time the menu button is depressed.

Start-up Processing: The '.suntools' File

Suntools can set up a standard arrangement of windows for the user when it starts up. It does this by reading the file *.suntools* in the user's home directory, and following the instructions there. A file name on the command line indicates an alternative file to be read. A *-n* switch suppresses start-up processing.

Each line in the file indicates data for one tool to be started. A line has the format:

path nl nt nw nh il it iw ih I args

where *path* is the file name of the tool to be run, *nl*, *nt*, *nw*, and *nh* are decimal numbers which define the window for the normal (open) tool, as left edge, top edge, width, and height; *il*, *it*, *iw*, and *ih* are the same for the iconic form of the tool; and *I* is a boolean flag which should be non-zero if the tool is to be started in its *iconic* form. The remainder of the line (*args*) is passed to the program running in the window as command-line arguments. The origin (0, 0) is the upper left hand corner of the screen. If a size parameter is equal to -1 then the Root Manager assigns an appropriate value.

For example, consider a *.suntools* file containing:

```
shelltool     0   72  -1  728      4   4  64  64      0     -c "vi ~/sked"
shelltool     372 0   -1  792      -1  -1 64  64      0     CONSOLE
clocktool     4   4   180 40       232 4  64  64      1
```

This starts *suntools* with three tools active:

- A Shell Tool in a window 38 by 80 characters, pushed into the lower left corner of the screen, leaving enough space above it to fit some icons, and running the editor *vi* on a file named 'sked' in the user's home directory. The string 'vi ~/sked' is passed as a command to be executed by the shell which runs in the tool. The icon is placed in the upper left corner of the screen.
- Another Shell Tool, extending nearly the full height of the screen (48 lines), and butted against the right edge. Its icon is placed by the system, and the shell waits for user typein; this shell's window will display all messages directed to the console.
- A Clock Tool starts out iconic, about a quarter of the way across the top of the screen.

The program *toolplaces* in the directory */usr/suntool* can be helpful in constructing the *.suntools* file; it writes (on the standard output) a description of the windows that exist at the time it is run.

Changing the System Font

The system has a default font built into it. It is possible to use another font instead by setting the environment variable DEFAULT_FONT to the name of the file containing the font to use. DEFAULT_FONT must have been set in the shell from which the program was invoked. A small number of valid alternative fonts can be found in the directory */usr/suntool/fixedwidthfonts*.

Multiple/Color Displays

The *suntools* program runs on either a monochrome or color screen. Each screen on a machine may have its own invocation of *suntools* running on it. The keyboard and mouse input devices are shared among multiple screens. The mouse cursor slides from one screen to another when the user moves the cursor off the edge of a screen. All the screens (frame buffers) supported by Sun Microsystems are supported.

A common configuration would be two screens, a monochrome and a color screen. Here is how a user could set up an instance of *suntools* on each screen.

1) Invoke suntools on the monochrome display by running "suntools". This starts *suntools* on the default frame buffer named */dev/fb*.

2) In a Shelltool, run "suntools -d /dev/cgone0". This starts *suntools* on a Sun-1 color screen.

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 the cursor moves off the right hand side of the monochrome screen. Similarly, the cursor slides from the color screen to the monochrome screen when the cursor moves off the left hand side of the color screen.

Getting Out

Any tool may be exited by invoking the Quit command in the Tool Manager menu while over the tool's window, as described above. Tools which run a shell (the Shell Tool and Graphics tool) also go away if that shell ever exits, so typing ^D to the shell in a terminal subwindow also makes the tool go away. Exit the whole window system by invoking the 'Exit' menu item in the Root Manager menu, and confirming that that's what you meant when the system inquires. It is wise to ensure that all windows are in a safe condition (for example, editors have written out all changes) first.

Suntools may also be exited via the keyboard by typing ^D followed by ^Q. There is no confirmation. This facility provides an escape hatch if the user inadvertently starts *suntools* when no mouse is attached to the system.

Initialization

Before *suntools* can be run, appropriate devices must be created for it in the */dev* directory. This need only be done once on each machine. If this has not yet been done,

- set your user id to *root*,
- change directory to */dev*, and
- execute the shell script 'MAKEDEV win0'.

Suntools also needs the file */etc/utmp* to have *read* and *write* permission for all users. It should have been installed with these permissions, but if not, you need to use *chmod* to change the permissions.

SEE ALSO

Other programs that run in the Suntools environment: *lockscreen*(1), *perf_mon*(1), and *adjacentscreens*(1).

FILES

/usr/suntool/suntools
~/.suntools
/usr/suntool/fixedwidthfonts/*
/usr/suntool/clocktool
/usr/suntool/gfxtool
/usr/suntool/panetool
/usr/suntool/shelltool
/usr/suntool/icontool
/usr/suntool/bouncedemo
/usr/suntool/framedemo
/usr/suntool/globeframes/*
/usr/suntool/jumpdemo
/usr/suntool/spheresdemo
/usr/suntool/toolplaces
/etc/utmp
/dev/win*x*
/dev/ptyp*x*

/dev/ttyp*x*
/dev/fb
/dev/kbd
/dev/mouse
/dev/MAKEDEV

BUGS

This release has the following notable bugs:

(1) Messages from the kernel ignore window boundaries unless console messages have been redirected, thus trashing the display. Recover from this by invoking the 'ReDisplay All' item on the Root Manager menu.

(2) Remote login to another machine should be done with a your terminal emulator subwindow matching the standard terminal size for the remote machine (for example, 34 by 80 character for a Sun workstation) The remote machine does not understand terminals with non-standard size.

(3) Acceptable performance on realistic tasks (for example editing with *vi* while running a C compile with *make*) requires about 600K of available user memory. With 1.1 Unix, the kernel may have to be reconfigured to make that space available. See the *System Managers Guide.*

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to ensure all disk writes have been completed before the processor is halted in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8), cron(8)

NAME

syslog – make system log entry

SYNOPSIS

syslog [**–p**] [**–t** *name*] [*–level*] [**–**] [*message ...*]

DESCRIPTION

Syslog sends the specified message (or *stdin* if **–** is specified) as a system log entry to the syslog daemon. The log entry is sent to the daemon on the machine specified by the *loghost* entry in the */etc/hosts* file.

OPTIONS

–p *Syslog* will log its process id in addition to the other information.

–t *name*
The specified name will be used as the "ident" for the log entry.

–level The message will be logged at the specified level. The level can be specified numerically, in the range 1 through 9, or symbolically using the names specified in the include file /usr/include/syslog.h (with the leading LOG_ stripped off). "*syslog* -HELP" will list the valid symbolic level names. Only the superuser can make log entries at levels less than or equal to SALERT.

– Each line of the standard input is sent as a log entry.

FILES

/usr/etc/in.syslog	syslog daemon
/usr/include/syslog.h	for names of logging levels

SEE ALSO

syslog(3), syslog(8)

NAME

tail – display the last part of a file

SYNOPSIS

tail [± *number* [**lbc**] [**fr**]] [file]

DESCRIPTION

Tail copies the named *file* to the standard output beginning at a designated place. If no file is named, the standard input is used.

OPTIONS

Options are all jammed together, not specified separately with their own – signs.

+*number*

Begin copying at distance + *number* from the beginning of the file. *Number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

–*number*

Begin copying at distance –*number* from the end of the file. *Number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

r Copy lines from the end of the file in reverse order. The default for **r** is to print the entire file this way.

f Follow the file as it grows, that is, don't quit at end of file, but rather wait and try to read repeatedly in hopes that the file will grow.

SEE ALSO

dd(1)

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length.

Various kinds of anomalous behavior may happen with character special files.

NAME

talk – talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

Talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
user@host

though *user@host* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, *talk* sends the message:

Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine

to the user you wish to talk to. At this point, the recipient of the message should reply by typing:

tutorial% **talk your_name@your_machine**

It doesn't matter from which machine the recipient replies, as long as their login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control-L redraws the screen, while your erase, kill, and word kill characters will work in *talk* as normal. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg*(1) command. At the outset talking is allowed. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent messy output.

FILES

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

SEE ALSO

mesg(1), who(1), mail(1), write(1)

NAME

tar – tape archiver

SYNOPSIS

tar –txruc[ovwfblmhpB0-9] [tarfile] [blocksize] file1 file2 ... **–C** dir file*n* ...

DESCRIPTION

Tar saves and restores multiple files on a single *tarfile* (usually a magnetic tape, but it can be any file). *Tar*'s actions are controlled by its first argument, the *key*, a string of characters containing exactly one function letter from the set **rxtuc** and one or more optional *function modifiers*. Other arguments to *tar* are file or directory names specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

FUNCTION LETTERS

r Write the named files on the end of the *tarfile*. Note that this option *does not work* with quarter-inch archive tapes.

x Extract the named files from the *tarfile*. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier versions.

t List the names of the specified files each time they occur on the *tarfile*. If no file argument is given, all of the names on the *tarfile* are listed.

u Add the named files to the *tarfile* if they are not there or have been modified since last put on the *tarfile*. Note that this option *does not work* with quarter-inch archive tapes.

c Create a new *tarfile* and write the named files onto it.

FUNCTION MODIFIERS

0,...,9

Select an alternate drive on which the tape is mounted. The default is drive 0 at 1600 bpi, which is normally */dev/rmt8*.

f Use the next argument as the name of the archive instead of */dev/rmt8*. If the name of the file is '–', *tar* writes to standard output or reads from standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain. *Tar* can also be used to copy hierarchies with the command:

```
tutorial% cd fromdir; tar cf – . | (cd todir; tar xfp –)
```

o Suppress information specifying owner and modes of directories which *tar* normally places in the archive. Such information makes former versions of *tar* generate an error message like:

'<name>/: cannot create'

when they encounter it.

v Normally *tar* does its work silently; the **v** (verbose) option displays the name of each file *tar* treats, preceded by the function letter. When used with the **t** function, **v** displays the *tarfile* entries in a form similar to **ls –l**.

w Wait for user confirmation before taking the specified action. If you use **w**, *tar* displays the action to be taken followed by the file name, and then waits for a 'y' response to proceed. No action is taken on the named file if you type anything other than a line beginning with 'y'.

b Use the next argument as the blocking factor for tape records. The default blocking factor is 20 blocks. The block size is determined automatically when reading tapes (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to the program at the time it runs. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

l Display error messages if all links to dumped files cannot be resolved. If **l** is not used, no error messages are printed.

m Do not restore modification times of extracted files. The modification time will be the time of extraction.

h Follow symbolic links as if they were normal files or directories. Normally, *tar* does not follow symbolic links.

p Restore the named files to their original modes, ignoring the present umask(2). Setuid and sticky information are also restored if you are the super-user. This option is only useful with the **x** key letter.

B Force *tar* to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that *tar* can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming.

I Ignore directory checksum errors.

If a file name is preceded by **–C** in a **c** (create) or **r** (replace) operation, *tar* will perform a *chdir*(2) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from */usr/include* and from */etc*, one might use:

```
tutorial% tar c –C /usr include –C /etc .
```

If you get a table of contents from the resulting *tarfile*, you will see something like:

```
include/
include/a.out.h
and all the other files in /usr/include
./
./chown
and all the other files in /etc
```

Note that the **–C** option only applies to *one* following directory name and *one* following file name.

EXAMPLES

Here is a simple example using *tar* to create an archive of your home directory onto */dev/rmt0*:

```
tutorial% cd                          position yourself in your home directory
tutorial% tar cvf /dev/rmt0 .         create the archive
          lots of messages from tar
tutorial%
```

The **c** option means create the archive; the **v** option makes *tar* tell you what it's doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (*/dev/rmt0* in this example).

Now you can read the table of contents from the archive like this:

```
tutorial% tar tvf /dev/rmt0           display table of contents of the archive
          lots of messages from tar
tutorial%
```

Where the **t** option is for displaying the table-of-contents of the archive. You can extract files from the archive like this:

```
tutorial% tar xvf /dev/rmt0           extract files from the archive
          lots of messages from tar
tutorial%
```

Where the **x** option is for extracting files from the archive.

Here is a note to clarify the effects of using *tar* to read from tapes where there are multiple archive files on the same tape. If there are multiple archive files on a tape, each is separated from the following one by an end-of-file marker. *Tar* does not read the end-of-file mark on the tape after it finishes reading an archive file because *tar* looks for a special header to decide when it has reached the end of the archive. Now if you try to use *tar* to read the next archive file from the tape, *tar* doesn't know enough to skip over the end-of-file mark and tries to read the end-of-file mark as an archive instead. The result of this is an error message from *tar* to the effect:

tar: blocksize=0

This means that to read another archive from the tape, the user must skip over the end-of-file marker before starting another *tar* command. You can achieve this via the *mt* command, as shown in the example below. Assume that you are reading from */dev/nrmt0.*

```
tutorial% tar xvfp /dev/nrmt0      read first archive from tape
          lots of messages from tar
tutorial% mt fsf 1  skip over the end-of-file marker
tutorial% tar xvfp /dev/nrmt0 read second archive from tape
          lots of messages from tar
tutorial%
```

Finally, here is a note on using *tar* to transfer files across the Ethernet. First, here is how to dump files from the local machine (tutorial) to a tape on a remote system (krypton):

```
tutorial% tar cvfb - 20 files | rsh krypton dd of=/dev/rmt0 obs=20b
          lots of messages from tar
tutorial%
```

In the example above, we are *creating* a *tarfile* with the **c** key letter, asking for *verbose output from tar* with the **v** option, specifying the name of the output *tarfile* via the **f** option (the standard output is where the *tarfile* appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the *tar* command *and* on the *dd* command.

Now, here is how to use *tar* to get files from a tape on the remote system (krypton) back to the local system (tutorial):

```
tutorial% rsh krypton dd if=/dev/rmt0 bs=20b | tar xvBfb - 20 files
          lots of messages from tar
tutorial%
```

In the example above, we are *extracting* from the *tarfile* with the **x** key letter, asking for *verbose output from tar* with the **v** option, specifying the name of the input *tarfile* via the **f** option (the standard input is where the *tarfile* appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option.

FILES

/dev/rmt?	half-inch magnetic tape interface
/dev/rar?	quarter-inch magnetic tape interface
/dev/rst?	SCSI tape interface
/tmp/tar*	

SEE ALSO

tar(5), cpio(1), dump(8), restore(8)

DIAGNOSTICS

Complains about bad key characters and tape read/write errors.
Complains if enough memory is not available to hold the link tables.

BUGS

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

There is no way to ask for the *n*-th occurrence of a file.

Tape errors are handled ungracefully.

The u option can be slow.

There is no way to selectively follow symbolic links.

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [–ms] [–mm] [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input *files* are copied to the standard output, except that lines between .TS and .TE command lines are assumed to describe tables and are reformatted. Details are given in the *tbl*(1) reference manual.

If no arguments are given, *tbl* reads the standard input, so *tbl* may be used as a filter. When *tbl* is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

OPTIONS

–ms Copy the –ms macro package to the front of the output file.

–mm Copy the –mm macro package to the front of the output file.

EXAMPLE

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

Household Population		
Town	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Branchburg	1644	3.49
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

troff(1), eqn(1)
Formatting Tables with tbl in
Editing and Text Processing on the Sun Workstation.

NAME

tee – copy standard output to many files

SYNOPSIS

tee [**–i**] [**–a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*.

OPTIONS

–i Ignore interrupts.

–a Append the output to the *files* rather than overwriting them.

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

telnet [host [port]]

DESCRIPTION

Telnet communicates with another host using the TELNET protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt ("telnet>"). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an *open* command (see below) with those arguments.

Once a connection has been opened, *telnet* enters input mode. In this mode, text typed is sent to the remote host. To issue *telnet* commands when in input mode, precede them with the *telnet* "escape character" (initially "^]"). When in command mode, the normal terminal editing conventions are available.

TELNET COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed.

open *host* [*port*]
Open a connection to the named host. If the no port number is specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*(5)) or an Internet address specified in the "dot notation".

close Close a TELNET session and return to command mode.

quit Close any open TELNET session and exit *telnet*.

z Suspend *telnet*. This command only works when the user is using the *csh*(1).

escape [*escape-char*]
Set the *telnet* "escape character". Control characters may be specified as "^" followed by a single letter; e.g. "control-X" is "^X".

status Show the current status of *telnet*. This includes the peer one is connected to, as well as the state of debugging.

options
Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations will be displayed. Options sent by *telnet* are displayed as "SENT", while options received from the TELNET server are displayed as "RCVD".

crmod Toggle carriage return mode. When this mode is enabled any carriage return characters received from the remote host will be mapped into a carriage return and a line feed. This mode does not affect those characters typed by the user, only those received. This mode is not very useful, but is required for some hosts that like to ask the user to do local echoing.

? [*command*]
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information available about the command only.

SEE ALSO

rlogin(1C)

BUGS

There is no provision in the standard TELNET protocol to support ^S/^Q type commands. This implementation is very simple because *rlogin*(1C) is the standard mechanism used to communicate locally with hosts.

NAME

test – condition command

SYNOPSIS

test expr

DESCRIPTION

test evaluates the expression *expr*, and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. *test* returns a non zero exit if there are no arguments.

The following primitives are used to construct *expr*.

–r file true if the file exists and is readable.

–w file true if the file exists and is writable.

–f file true if the file exists and is not a directory.

–d file true if the file exists exists and is a directory.

–s file true if the file exists and has a size greater than zero.

–t [fildes]
true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

–z s1 true if the length of string *s1* is zero.

–n s1 true if the length of the string *s1* is nonzero.

s1 **==** s2 true if the strings *s1* and *s2* are equal.

s1 **!=** s2 true if the strings *s1* and *s2* are not equal.

s1 true if *s1* is not the null string.

n1 **–eq** n2
true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **–ne**, **–gt**, **–ge**, **–lt**, or **–le** may be used in place of **–eq**.

These primaries may be combined with the following operators:

! unary negation operator

–a binary *and* operator

–o binary *or* operator

(expr **)**
parentheses for grouping.

–a has higher precedence than **–o**. Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO

sh(1), find(1)

NAME

time – time a command

SYNOPSIS

time [command]

DESCRIPTION

There are two distinct versions of *time*: it is built in to the C-shell, and is an executable program available in */bin/time* when using the Bourne shell. In both cases, times are displayed on the diagnostic output stream.

In the case of the C-shell, a *time* command with no *command* argument simply displays a summary of time used by this shell and its children. When arguments are given the specified simple *command* is timed and the C-shell displays a time summary as described below under the *time* variable. If necessary, an extra shell is created to print the time statistic when the command completes.

The default resource-usage summary is a line of the form:

*uuu.uu sss.ss ee:ee pp% xxx+ ddd*k *iii+ ooo*io *mmm*pf+ *ww*w

where *uuu.u* is the user time (U), *sss.s* is the system time (S), *ee:ee* is the elapsed time (E), *pp* is the percentage of CPU time versus elapsed time (P), *xxx* is the average shared memory in Kilobytes (X), *ddd* is the average unshared data space in Kilobytes (D), *iii* and *ooo* are the number of block input and output operations respectively (I and O), *mmm* is the number of page faults (M), and *ww* is the number of swaps (W).

The **time** variable controls the display that the C-shell prints when it times a command. The **time** variable can be supplied with one or two values. The first value is a number — *n* for instance. The C-shell displays a resource-usage summary for any command running for more than *n* CPU seconds. The second value is optional and is a character string which determines which resources the user wishes displayed. The character string can be any string of text with embedded control key-letters in it. A control key-letter is a percent sign (**%**) followed by a single *upper-case* letter. To print a percent sign, use two percent signs in a row. Unrecognized key-letters are simply printed. The control key-letters are:

D	Average amount of unshared data space used in Kilobytes.
E	Elapsed (wallclock) time for the command.
F	Page faults.
I	Number of block input operations.
K	Average amount of unshared stack space used in Kilobytes.
M	Maximum real memory used during execution of the process.
O	Number of block output operations.
P	Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time.
S	Number of seconds of CPU time consumed by the kernel on behalf of the user's process.
U	Number of seconds of CPU time devoted to the user's process.
W	Number of swaps.
X	Average amount of shared memory used in Kilobytes.

The *time* command in '/bin/time' times the given *command*, which must be specified, that is, *command* is not optional as it is in the C-shell's timing facility. When the command is complete, *time* displays the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

EXAMPLES

The two examples here show the differences between the *csh* version of *time* and the version in */bin/time*. The example assumes that *csh* is the shell in use.

```
angel% time wc /usr/man/man1/csh.1
    1876   11223   65895 /usr/man/man1/csh.1
```

```
2.7u 0.9s 0:03 91% 3+ 5k 19+ 2io 1pf+ 0w
angel% /bin/time wc /usr/man/man1/csh.1
  1876  11223  65895 /usr/man/man1/csh.1
     4.3 real         2.7 user         1.0 sys
angel%
```

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 50th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tip, cu – connect to a remote system

SYNOPSIS

tip [**–v**] [*–speed*] system-name
tip [**–v**] [*–speed*] phone-number
cu phone-number [**–t**] [**–s** *speed*] [**-a** *acu*] [**–l** *line*] [**–#**]

DESCRIPTION

Tip and *cu* establish a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote computer. It goes without saying that you must have a account on the machine (or equivalent) to which you wish to connect. The preferred interface is *tip*. The *cu* interface is included for those people attached to the 'call UNIX' command of version 7. This manual page describes only *tip*.

When *tip* starts up it reads commands from the file *.tiprc* in your home directory. If you use the **–v** option on the *tip* command line, *tip* displays these commands as it executes them. See the discussion on *variables* later on.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde ("~") appearing as the first character of a line is an escape signal; the following are recognized:

~^D ~. Drop the connection and exit (you may still be logged in on the remote machine).

~c [*name*] Change directory to name (no argument implies change to your home directory).

~! Escape to a shell (exiting the shell will return you to *tip*).

~> Copy file from local to remote.

~< Copy file from remote to local.

~p *from* [*to*]
Send a file to a remote UNIX host. When you use the put command, the remote UNIX system runs the command string
cat > *to*
while *tip* sends it the *from* file. If the *to* file isn't specified, the *from* file name is used. This command is actually a UNIX specific version of the "~>" command.

~t *from* [*to*]
Take a file from a remote UNIX host. As in the put command the *to* file defaults to the *from* file name if it isn't specified. The remote host executes the command string
cat > *from*; **echo ^A**
to send the file to *tip*.

~| Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.

~C Connect a program to the remote machine. The command string sent to the program is processed by the shell. The program inherits file descriptors 0 as remote line input, 1 as remote line output, and 2 as tty standard error.

~# ~# Send a BREAK to the remote system. For systems which don't support the necessary *ioctl* call the break is simulated by a sequence of line speed changes and DEL characters.

~s Set a variable (see the discussion below).

~^Z Stop *tip* (only available when run under the C-Shell).

~? Get a summary of the tilde escapes

Copying files requires some cooperation on the part of the remote host. When a ~> or ~< escape is used to send a file, *tip* prompts for a file name (to be transmitted or received) and a command to be sent to the remote system, in case the file is being transferred from the remote system. The default end of transmission string for transferring a file from the local system to the remote is specified in the *remote*(5) file, but may be changed by the set command. While *tip* is transferring a file the number of lines transferred will be continuously displayed on the screen. A file transfer may be aborted with an interrupt. An example of the dialogue used to transfer files is given below (input typed by the user is shown in bold face).

```
arpa% tip monet
[connected]
...(assume we are talking to another UNIX system)...
ucbmonet login: sam
Password:
monet% cat > foo.c
~> Filename: foo.c
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~< Filename: reply.c
List command for remote host: cat reply.c
65 lines transferred in 2 minutes
monet%
...(or, equivalently)...
monet% ~p foo.c
...(actually echoes as ~[put] foo.c)...
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~t reply.c
...(actually echoes as ~[take] reply.c)...
65 lines transferred in 2 minutes
monet%
...(to print a file locally)...
monet% ~|Local command: pr -h foo.c | lpr
List command for remote host: cat foo.c
monet% ~^D
[EOT]
...(back on the local system)...
```

The *remote*(5) file contains the definitions for remote systems known by *tip*; refer to the remote manual page for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, for example:

```
tip -300 mds
```

When *tip* establishes a connection it sends out a connection message to the remote system. The default value for this string may be found in the remote file.

At any time that *tip* prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote machine.

When *tip* attempts to connect to a remote system, it opens the associated device with an exclusive-open *ioctl*(2) call. Thus only one user at a time may access a device. This is to prevent multiple processes from sampling the terminal line. In addition, *tip* honors the locking protocol used by *uucp*(1C).

AUTO-CALL UNITS

Tip may be used to dial up remote systems using a number of auto-call unit's (ACU's). When the remote system description contains the "du" attribute, tip will use the call-unit ("cu"), ACU type ("at"), and phone numbers ("pn") supplied. Normally tip will print out verbose messages as it dials. See *remote*(5) for details of the remote host specification.

Depending on the type of auto-dialer being used to establish a connection the remote host may have garbage characters sent to it upon connection. The user should never assume that the first characters typed to the foreign host are the first ones presented to it. The recommended practice is to immediately type a "kill" character upon establishing a connection (most UNIX systems support "@" as the initial kill character).

Tip currently supports the Ventel MD-212+ autodialer modem.

REMOTE HOST DESCRIPTIONS

Descriptions of remote hosts are normally located in the system-wide file */etc/remote*. However, a user may maintain personal description files (and phone numbers) by defining and exporting the shell variable REMOTE. The *remote* file must be readable by *tip*, but a secondary file describing phone numbers may be maintained readable only by the user. This secondary phone number file defaults to */etc/phones*, unless the shell variable PHONES is defined and exported. As described in *remote*(5), the *phones* file is read when the host description's phone number(s) capability is an "@". The phone number file contains lines of the form:

```
system-name    phone-number
```

Each phone number found for a system is tried until either a connection is established, or an end of file is reached. Phone numbers are constructed from "0123456789-=*", where the "=" and "*" are used to indicate a second dial tone should be waited for (ACU dependent).

VARIABLES

Tip maintains a set of variables which are used in normal operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the "s" escape. The syntax for variables is patterned after *vi*(1) and *mail*(1). Supplying "all" as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a "?" to the end. For example "escape?" displays the current escape character.

Variables are numeric, string, character, or boolean values. Boolean variables are set merely by specifying their name. They may be reset by prepending a "!" to the name. Other variable types are set by appending an "=" and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run time by placing set commands (without the "~s" prefix in a file *.tiprc* in one's home directory). The –v option causes *tip* to display the sets as they are made.

Finally, the variable names must either be completely specified or an abbreviation may be given. The following list details those variables known to *tip*, their abbreviations (surrounded by brackets), and their default values. Those variables initialized from the remote file are marked with a "*". A mode is given for each variable; capitalization indicates the read or write capability is given only to the super-user.

Variable	Type	Mode	Default	Description
[be]autify	bool	rw	true	discard unprintables when scripting
[ba]udrate	num	rW	*	connection baud rate
[dial]timeout	num	rW	60	timeout (seconds) when establishing connection
[eofr]ead	str	rw	*	char's signifying EOT from the remote host
[eofw]rite	str	rw	*	string sent for EOT

[eol]	str	rw	*	end of line indicators
[es]cape	char	rw	'~'	command prefix character
[ex]ceptions	str	rw	"\t\n\f\b"	char's not discarded due to beautification
[fo]rce	char	rw	'^P'	force character
[fr]amesize	num	rw	*	size of buffering between writes on reception
[ho]st	str	r	*	name of host connected to
[lock]	str	RW	/tmp/aculock	lock file for ACU logging
[log]	str	RW	/usr/adm/aculog	ACU log file
[phones]	str	r	/etc/phones	file for hidden phone numbers
[pr]ompt	char	rW	'\n'	end of line indicator set by host
[ra]ise	bool	rw	false	upper case mapping switch
[r]aise[c]har	char	rw	'^A'	interactive toggle for raise
[rec]ord	str	rw	"tip.record"	name of script output file
[remote]	str	r	/etc/remote	system description file
[sc]ript	bool	rw	false	session scripting switch
[tab]expand	bool	rw	false	expand tabs during file transfers
[verb]ose	bool	rw	true	make noise during file transfers
[SHELL]	str	rw	"/bin/sh"	name of shell for ~! escape
[HOME]	str	rw	""	home directory for ~c escape

ENVIRONMENT VARIABLES

The following variables are read from the environment:

REMOTE The location of the *remote* file.

PHONES The location of the file containing private phone numbers.

HOST A default host to connect to.

HOME One's log-in directory (for chdirs).

SHELL The shell to fork on a "~!" escape.

FILES

~/.tiprc initialization file.
/usr/spool/uucp/LCK..* lock file to avoid conflicts with *uucp*

DIAGNOSTICS

Diagnostics are, hopefully, self explanatory.

SEE ALSO

remote(5), phones(5)

NAME

touch – update date last modified of a file

SYNOPSIS

touch [–c] [–f] file ...

DESCRIPTION

Touch attempts to set the modified date of each *file*. If the file exists, this is done by reading a character from the file and writing it back.

Touch is valuable when used in conjunction with *make*(1), where, for instance, you might want to force a complete rebuild of a program composed of many pieces. In such a case, you might type, for example:

```
% touch *.c
% make
```

and the *make* would then see that all the .c files were more up to date than all the corresponding .o files, and would start the build from scratch.

OPTIONS

–c Do not attempt to create a *file* if it does not exist.

–f Attempt to force the touch in spite of read and write permissions on a *file*.

SEE ALSO

utimes(2)

NAME

tr – translate characters

SYNOPSIS

tr [**–cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. The arguments *string1* and *string2* are considered sets of characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character.

In either string the notation *a–b* means a range of characters from *a* to *b* in increasing ASCII order. The character '\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A '\' followed by any other character stands for that character.

OPTIONS

Any combination of the options **–cds** may be used:

–c Complement the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal;

–d Delete all input characters in *string1;*

–s Squeeze all strings of repeated output characters that are in *string2* to single characters.

EXAMPLE

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr –cs A–Za–z '\012' <file1 >file2
```

SEE ALSO

ed(1), ascii(7), expand(1)

BUGS

Won't handle ASCII NUL in *string1* or *string2;* always deletes NUL from input.

NAME

troff – typeset or format documents

SYNOPSIS

troff [**-o***pagelist*] [**-n***N*] [**-s***N*] [**-m** *name*] [**-r***aN*] [**-l**] [**-q**] [**-t**] [**-f**] [**-w**] [**-b**] [**-a**] [**-p***N*] [file] ...

DESCRIPTION

Troff formats text in the named *files*. The output is by default destined for printing on a Graphic Systems C/A/T phototypesetter, but suitable postprocessing software can convert the C/A/T output to a form which can be directed to other high-resolution devices. See also the *nroff*(1) manual page, which describes a formatter which formats text for typewriter-like devices. The capabilities of both *troff* and *nroff* are described in *Formatting Documents with Nroff and Troff*.

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input.

OPTIONS

Options may appear in any order so long as they appear before the *files*.

-o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial -*N* means from the beginning to page *N*; and a final *N*- means from *N* to the end.

-n*N* Number first generated page *N*.

-m*name*
Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

-r*aN* Set register *a* (one-character) to *N*.

-l Read standard input after the input files are exhausted.

-q Invoke the simultaneous input-output mode of the **rd** request.

-t Direct output to the standard output instead of the phototypesetter. In general, you will have to use this option if you don't have a typesetter attached to the system.

-a Send a printable ASCII approximation of the results to the standard output.

Some options of *troff* only apply if you have a C/A/T typesetter attached to your system. These options are here for historical reasons:

-s*N* Stop every *N* pages. *Troff* stops the phototypesetter every *N* pages, produces a trailer to allow changing cassettes, and resumes when the typesetter's start button is pressed.

-f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

-w Wait until phototypesetter is available, if currently busy.

-b Report whether the phototypesetter is busy or available. No text processing is done.

-pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elasped time.

If the file */usr/adm/tracct* is writable, *troff* keeps phototypesetter accounting records there. The integrity of that file may be secured by making *troff* a 'set user-id' program.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

SEE ALSO

Formatting Documents with Nroff and Troff in
Editing and Text Processing on the Sun Workstation
nroff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

True **and** ***false*** **are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.**

EXAMPLE

```
while true
do
    command list
done
```

SEE ALSO

csh(1), sh(1), false(1)

DIAGNOSTICS

True **has exit status zero.**

NAME

tset – establish terminal characteristics for the environment

SYNOPSIS

tset [**–ec**] [**–kc**] [**–**] [**–n**] [**–I**] [**–Q**] [**–m** [ident][test baudrate]:type] ... [type]

reset ...

DESCRIPTION

Tset sets up your terminal when you first log in to a UNIX system. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. *Tset* first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each UNIX port is specified in the */etc/ttytype* database. Type names for terminals may be found in the *termcap*(5) database. If a port is not wired permanently to a specific terminal (not hardwired) it is given an appropriate generic identifier such as *dialup*.

When no arguments are specified, *tset* simply reads the terminal type out of the TERM environment variable and re-initializes the terminal. The rest of this manual entry deals with mode and environment initialization — typically done once at login — and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script (*.profile* for *sh*(1) users or *.login* for *csh*(1) users) it is desirable to give information about the type of terminal you will usually use on ports which are not hardwired. These ports are identified in */etc/ttytype* as *dialup* or *plugboard* or *arpanet*, etc. To specify what terminal type you usually use on these ports, the **–m** (map) option flag is followed by the appropriate port type identifier, an optional baud rate specification, and the terminal type. (The effect is to "map" from some conditions to a terminal type, that is, to tell *tset* "If I'm on this kind of port, guess that I'm on that kind of terminal".) If more than one mapping is specified, the first applicable mapping prevails. A missing port type identifier matches all identifiers. Any of the alternate generic names given in *terminfo* may be used for the identifier.

A *baudrate* is specified as with *stty*(1), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate *test* may be any combination of: **>**, **@**, **<**, and **!**; **@** means "at" and **!** inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **–m** within "'" (apostrophe) characters; users of *csh*(1) must also put a "\" before any "!" used here.

Thus, the command

tset –m 'dialup>300:adm3a' -m dialup:dw2 -m 'plugboard:?adm3a'

sets the terminal type to *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a *dw2* if the port is (otherwise) a dialup (that is, at 300 baud or less). If the *type* finally determined by *tset* begins with a question mark, the user is asked if s/he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Thus, in the above case, the user is queried on a plugboard port as to whether s/he is actually using an *adm3a*.

If no mapping applies and a final *type* option, not preceded by a **–m**, is given on the command line then that type is used; otherwise the identifier found in the */etc/ttytype* database is used as the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by *tset*, and information about the terminal's capabilities to a shell's environment. This can be done using the **–** option; using the Bourne shell, *sh*(1):

export TERM; TERM=`tset – *options...*`

or using the C shell, *csh*(1):

setenv TERM `tset – *options...*`

With *csh* it is convenient to make an alias in your .cshrc:

```
alias tset 'setenv TERM `tset - \!*`'
```

Either of these aliases allow the command

```
tset 2621
```

to be invoked at any time from your login csh. **"Note to Bourne Shell users:"** It is not possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. If a process could set its parent's environment, none of this nonsense would be necessary in the first place.

Once the terminal type is known, *tset* engages in terminal driver mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and newline expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ('#' on standard systems), the erase character is changed to BACKSPACE (Control-H).

If **tset** is invoked as **reset**, it will set cooked and echo modes, turn off cbreak and raw modes, turn on newline translation, and restore special characters to a sensible state before any terminal dependent processing is done. Any special character that is found to be NULL or "-1" is reset to its default value.

This is most useful after a program dies leaving a terminal in a funny state. You may have to type "<LF>reset<LF>" to get it to work since <CR> may not work in this state. Often none of this will echo.

EXAMPLES

These examples all assume the Bourne shell and use the - option. If you use *csh*, use one of the variations described above. Note that a typical use of *tset* in a .profile or .login will also use the **-e** and **-k** options, and often the **-n** or **-Q** options as well. These options have not been included here to keep the examples small. **(NOTE:** some of the examples given here appear to take up more than one line, for text processing reasons. When you type in real *tset* commands, you must enter them entirely on one line.)

At the moment, you are on a 2621. This is suitable for typing by hand but not for a .profile, unless you are *always* on a 2621.

```
export TERM; TERM=`tset - 2621`
```

You have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttytype.

```
export TERM; TERM=`tset - -m dialup:h19`
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use a vt100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone else's terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
export TERM; TERM=`tset - -m 'switch>1200:?vt100' -m 'switch<=1200:2621'
```

All of the above entries will fall back on the terminal type specified in */etc/ttytype* if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
export TERM; TERM=`tset - ?adm3a`
```

If the file */etc/ttytype* is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
export TERM; TERM=`tset - -m '>1200:vt100' 2621`
```

Here is a fancy example to illustrate the power of *tset* and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in */etc/ttytype*. You want your erase character set to control H, your kill character set to control U, and don't want *tset* to print the "Erase set to Backspace, Kill set to Control U" message.

```
export TERM; TERM=`tset -e -k^U -Q - -m 'switch<=1200:concept100' -m
'switch:?vt100' -m dialup:concept100 -m arpanet:dm2500`
```

OPTIONS

-e*c* set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually ^H. The character *c* can either be typed directly, or entered using the hat notation used here.

-k*c* is similar to **-e** but for the line kill character rather than the erase character; *c* defaults to ^U (for purely historical reasons). The kill characters is left alone if **-k** is not specified. The hat notation can also be used for this option.

- The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the TERM environment variable.

-n On systems with the Berkeley 4BSD tty driver, specifies that the new tty driver modes should be initialized for this terminal. For a CRT, the CRTERASE and CRTKILL modes are set only if the baud rate is 1200 or greater.

-I suppresses transmitting terminal-initialization strings.

-Q suppresses printing the "Erase set to" and "Kill set to" messages.

FILES

/etc/ttytype	port name to terminal type mapping database
/etc/termcap	terminal capability database
/usr/lib/tabset/*	tab setting sequences for various terminals. Pointed to by *termcap* entries.

SEE ALSO

csh(1), sh(1), stty(1), ttytype(5), termcap(5), environ(5)

BUGS

The *tset* command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the *login*(1) program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty – get terminal name

SYNOPSIS

tty [–s]

DESCRIPTION

Tty **prints the pathname of the user's terminal unless the –s (silent) option is given. In either case, the exit value is zero if the standard input is a terminal, and one if it is not.**

NAME

ul – do underlining

SYNOPSIS

ul [**–i**] [**–t** *terminal*] [file ...]

DESCRIPTION

Ul reads the named *files* (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. *ul* uses the */etc/termcap* file to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

OPTIONS

–t Override the terminal kind specified in the environment. If the terminal cannot underline, underlining is ignored.

–i Indicate underlining onto by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

BUGS

Nroff usually generates a series of backspaces and underlines intermixed with the text to indicate underlining. *Ul* makes attempt to optimize the backward motion.

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

/usr/sccs/unget [**–r** *SID*] [**–s**] [**–n**] file ...

DESCRIPTION

Unget undoes the effect of a **get –e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

OPTIONS

Options apply independently to each named file.

–r *SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The **–r** option is necessary only if two or more outstanding *gets* for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary but omitted from the command line.

–s Suppress displaying the intended delta's *SID*.

–n Retain the gotten file — it is normally removed from the current directory.

SEE ALSO

sccs(1), delta(1), get(1), sact(1).
Source Code Control System in *Programming Tools for the Sun Workstation.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [–udc [+n] [–n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see *sort*(1).

OPTIONS

–u Copy only those lines which are *not* repeated in the original file.

–d Write one copy of just the repeated lines.

The normal output of *uniq* is the union of the **–u** and **–d** options

–c supersedes **–u** and **–d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

You have: inch
You want: cm
* *2.54000e+00*
/ *3.93701e-01*

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

You have: 15 pounds force/in2
You want: atm
* *1.02069e+00*
/ *9.79730e-01*

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgium-franc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

BUGS

Don't base your financial plans on the currency conversions.

NAME

uptime – show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w*(1) command.

EXAMPLE

```
angel% uptime
 6:47am  up 6 days, 16:38,  1 users,  load average: 0.69, 0.28, 0.17
angel%
```

FILES

/vmunix system name list

SEE ALSO

w(1)

NAME

users – compact list of users who are on the system

SYNOPSIS

users

DESCRIPTION

Users lists the login names of the users currently on the system in a compact, one-line format:

```
% users
john paul george
%
```

FILES

/etc/utmp

SEE ALSO

who(1)

NAME

uucp, uulog – unix to unix copy

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form

system-name!pathname

where 'system-name' is taken from a list of system names which *uucp* knows about. Shell metacharacters ? * [ ] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~*user*; where *user* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

UUCP OPTIONS

The following options are interpreted by *uucp*.

–d Make all necessary directories for the file copy.

–c Use the source file when copying out rather than copying the file to the spool directory.

–m Send mail to the requester when the copy is complete.

UULOG OPTIONS

Uulog maintains a summary log of *uucp* and *uux*(1C) transactions in the file */usr/spool/uucp/LOGFILE* by gathering information from partial log files named */usr/spool/uucp/LOG.*.?*. It removes the partial log files.

The options cause *uulog* to print logging information:

–s*sys* Print information about work involving system *sys*.

–u*user* Print information about work done for the specified *user*.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/*	other data and program files

SEE ALSO

uux(1C), mail(1)

Uucp Implementation Description in the *Sun System Manager's Manual.*

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

BUGS

All files received by *uucp* will be owned by *uucp*.
The –m option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [ ] will not activate the –m option.

NAME

uuencode,uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [source-file] destination-file | **mail** sys1!sys2!...!decode
uudecode [source-file]

DESCRIPTION

Uuencode and *uudecode* are used to send a binary file via uucp (or other) mail. This combination of commands can be used over indirect mail links even when *uusend*(1C) is not available.

Uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

Uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user "decode" should be filtered through the *uudecode* program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using *sendmail* or by making *rmail* be a link to *Mail* instead of *mail*. In each case, an alias must be created in a master file to get the automatic invocation of *uudecode*.

If these facilities are not available, the file can be sent to a user on the remote machine who can *uudecode* it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

SEE ALSO

uuencode(5), uusend(1C), uucp(1C), uux(1C), mail(1)

BUGS

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking *uudecode* (often *uucp)* must have write permission on the specified file.

NAME

uusend – send a file to a remote host

SYNOPSIS

uusend [–m mode] sourcefile sys1!sys2!..!remotefile

DESCRIPTION

Uusend **sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of** ***uucp*****(1C) links needs to connect the two systems.**

The sourcefile can be "–", meaning to use the standard input. Both of these options are primarily intended for internal use of ***uusend*****.**

The remotefile can include the ~userid syntax.

OPTIONS

–m ***mode***

Take the mode of the file on the remote end from the octal number specified as ***mode*****. The mode of the input file is used if the –m option is not specified.**

DIAGNOSTICS

If anything goes wrong any further away than the first system down the line, you will never hear about it.

SEE ALSO

uux(1C), uucp(1C), uuencode(1C)

BUGS

This command shouldn't exist, since ***uucp*** **should handle it.**

All systems along the line must have the ***uusend*** **command available and allow remote execution of it.**

Some UUCP systems have a bug where binary files cannot be the input to a ***uux*** **command. If this bug exists in any system along the line, the file will show up severely munged.**

NAME

uux – unix to unix command execution

SYNOPSIS

uux [–] command-string

DESCRIPTION

Uux will gather 0 or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

The command-string is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by system-name!. A null system-name is interpreted as the local system.

File names may be one of

(1) a full pathname;

(2) a pathname preceded by ~*xxx*; where *xxx* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

The '–' option will cause the standard input to the *uux* command to be the standard input to the command-string.

For example, the command

uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"

will get the f1 files from the usg and pwba machines, execute a *diff* command and put the results in f1.diff in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

FILES

/usr/spool/uucpspool directory
/usr/lib/uucp/* other data and programs

SEE ALSO

uucp(1C)
D. A. Nowitz, *Uucp Implementation Description*

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from *uux*. Typically, a restricted site will permit little other than the receipt of mail via *uux*.

BUGS

Only the first command of a shell pipeline may have a system-name!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do.
The shell tokens << and >> are not implemented.
There is no notification of denial of execution on the remote machine.

NAME

val – validate SCCS file

SYNOPSIS

/usr/sccs/val –
/usr/sccs/val [**–s**] [**–r** *SID*] [**–m** *name*] [**–y** *type*] file ...

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order.

Val has a special argument, –, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

OPTIONS

Options apply independently to each named file on the command line.

–s Silence diagnostic messages normally generated for errors detected while processing the specified files.

–r *SID* The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (for instance, **r1** is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (for instance, **r1.0** or **r1.1.0** are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

–m *name*
name is compared with the SCCS %M% keyword in *file*.

–y *type* *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, that is, can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate option;
bit 2 = corrupted SCCS file;
bit 3 = can't open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **–y** mismatch;
bit 7 = %M%, **–m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned — logical OR of the codes generated for each command line and file processed.

SEE ALSO

sccs(1), **admin**(1), **delta**(1), **get**(1), **prs**(1).
Source Code Control System in *Programming Tools for the Sun Workstation*.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vax – is current machine a vax

SYNOPSIS

if vax; then ...; fi *(sh)*

if { vax } then *(csh)*
...
endif

DESCRIPTION

The *vax* command is, on VAX'en, the same as *true*(1); on Sun Workstations and other machines it is the same as *false*(1).

SEE ALSO

false(1), sun(1), true(1)

NAME

vgrind – grind nice listings of programs

SYNOPSIS

vgrind [**–f**] [**–**] [**–t**] [**–n**] [**–x**] [**–W**] [**–sn**] [**–h** header] [**–d** file] [**–l***language*] file ...

DESCRIPTION

Vgrind formats the program sources which are *file* arguments in a nice style using *troff*(1). Comments are placed in italics, keywords in bold face, and the name of the current function is listed down the margin of each page as it is encountered.

Vgrind runs in two basic modes, filter mode or regular mode. In filter mode *vgrind* acts as a filter in a manner similar to *tbl*(1). The standard input is passed directly to the standard output except for lines bracketed by the *troff*-like macros:

.vS - starts processing

.vE - ends processing

These lines are formatted as described above. The output from this filter can be passed to *troff* for output. There need be no particular ordering with *eqn*(1) or *tbl*(1).

In regular mode *vgrind* accepts input *files*, processes them, and passes them to *troff*(1) for output.

In both modes *vgrind* passes any lines beginning with a decimal point without conversion.

OPTIONS

–f force filter mode.

– take from standard input (default if **–f** is specified).

–t similar to the same option in *troff*, that is, formatted text goes to the standard output.

–n do not make keywords boldface.

–x output the index file in a 'pretty' format. The index file itself is produced whenever *vgrind* is run with a file called *index* in the current directory. The index of function definitions can then be run off by giving *vgrind* the **–x** option and the file *index* as argument.

–W force output to the (wide) Versatec printer rather than the (narrow) Varian.

–s specifies a point size to use on output (exactly the same as the argument of a *troff* .ps (point size) request.

–h specifies a particular header to put on every output page (default is the file name).

–d specifies an alternate language definitions file (default is */usr/lib/vgrindefs*).

–l specifies the language to use. Currently known are PASCAL (**–lp**), C (**–lc** the default), CSH (**–lcsh**), SHELL (**–lsh**), RATFOR (**–lr**), and ICON (**–li**).

FILES

index	file where source for index is created
/usr/lib/tmac/tmac.vgrind	macro package
/usr/lib/vfontedpr	preprocessor
/usr/lib/vgrindefs	language descriptions

BUGS

Vfontedpr assumes that a certain programming style is followed:

For C – function names can be preceded on a line only by spaces, tabs, or an asterisk. The parenthesized arguments must also be on the same line.

For PASCAL – function names need to appear on the same line as the keywords *function* or *procedure*.

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of spaces to align source code fails miserably; if you plan to *vgrind* your program you should use tabs. This is somewhat inevitable since the font used by *vgrind* is variable width.

The mechanism of *ctags*(1) in recognizing functions should be used here.

NAME

vi – screen oriented (visual) display editor based on ex

SYNOPSIS

vi [**–t** tag] [**–r**] [**+***command*] [**–l**] [**–w***n*] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*(1). *Ex* and *vi* are in fact the same text editor; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

FILES

See *ex*(1).

SEE ALSO

ex (1), edit (1), "Vi Quick Reference" card,
Using vi, the Visual Display Editor in
Editing and Text Processing on the Sun Workstation.

BUGS

Software tabs using ^T work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Repeating a change which wraps over the margin when *wrapmargin* is in effect doesn't generally work well: sometimes it just makes a mess of the change, and sometimes even leaves you in insert mode. A way to work around the problem is to replicate the changes using yank and put.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open.*

NAME

vi – view a file without changing it using the vi visual editor

SYNOPSIS

view [–t tag] [–r] [+*command*] [–l] [–w*n*] name ...

DESCRIPTION

View **uses the *vi* (visual) or display oriented text editor to browse through a file interactively without actually making any changes to the file. It is possible to get to the command mode of *ex* from within *view* and vice-versa, just as when using *vi*.**

FILES

See *ex*(1).

SEE ALSO

ex (1), edit (1), "Vi Quick Reference" card,
Using vi, the Visual Display Editor **in**
Editing and Text Processing on the Sun Workstation.

NAME

vplot – plot graphics on the Versatec

SYNOPSIS

vplot [**–W**] [**–V**] [**–b** *lpr-arg*] file

DESCRIPTION

Vplot reads *plot*(5) format graphics input from the file specified by *file* (standard input if no *file* is specified) and produces a plot on the Varian or Versatec.

OPTIONS

–W force output to the (wide) Versatec printer rather than the standard Versatec printer.

–V force output to the standard Versatec printer.

–b *lpr-arg*

arg (the next argument on the command line) specifies extra arguments to *lpr*(1).

SEE ALSO

plot(1G), lpr(1), plot(5) Reads standard graphics input and produces a plot on the Varian or Versatec

NAME

vswap – convert a foreign font file

SYNOPSIS

vswap [**-r**]

DESCRIPTION

Without the **-r** option *vswap* translates its standard input (which must be a *vfont* file in the reversed byte order) into a locally correct vfont file on its standard output. With the **-r** option *vswap* translates its standard input (which must be a *vfont* file in the local byte order) into a byte-reversed vfont file on its standard output.

The UNIX *vfont* representation for fonts is a binary file containing machine-dependent elements – short (16-bit) integers, in particular. There are (at least) two common ways of representing a 16-bit integer. A program compiled on a VAX will expect the VAX format, while the same program compiled on a Motorola 68000 will expect 68000 format. *Vswap* can be used to convert font files created on a VAX to the format required to use them on the Sun. It can also convert Sun-format font files to VAX format (with the **-r** option). Since the font files are in the byte order of the local machine, programs which access the font files don't need to be concerned with byte-swapping issues. This could be considered a bug.

SEE ALSO

troff(1), vfont(5)

BUGS

A machine-independent font format should be defined.

NAME

w – who is on and what they are doing

SYNOPSIS

w [–h] [–s] [user]

DESCRIPTION

W displays a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields displayed are: the users login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

If a *user* name is included, output is restricted to that user.

OPTIONS

–h Suppress the heading.

–s Produce a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands.

–l Produce a long form of output, which is the default.

EXAMPLE

```
angel% w
 7:36am  up 6 days, 16:45,  1 users,  load average: 0.20, 0.23, 0.18
User     tty      login@  idle   JCPU   PCPU  what
henry    console  7:10am     1   10:05   4:31  w
angel%
```

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), ps(1), utmp(5)

BUGS

The notion of the "current process" is muddy. The current algorithm is "the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal". This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, *w* prints "–".)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is "charged" with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detecting background jobs. It will sometimes find a background job instead of the right one.

NAME

wait – await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & or bg have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the Shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1), csh(1)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for. (This bug does not apply to *csh*(1).)

NAME

wall – write to all users

SYNOPSIS

wall [file]

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

NAME

wc – word count

SYNOPSIS

wc [–lwc] [file ...]

DESCRIPTION

Wc counts lines, words, and characters in the named *files*, or in the standard input if no *file* names appear. A word is a string of characters delimited by spaces, tabs, or newlines.

OPTIONS

If an argument beginning with one of 'lwc' is present, the specified counts are selected by the letters:

l Count lines.

w Count words.

c Count characters.

The default is –lwc (count lines, words, and characters).

EXAMPLE

```
angel% wc /usr/man/man1/{csh.1,sh.1,telnet.1}
    1876   11223   65895  /usr/man/man1/csh.1
     674    3310   20338  /usr/man/man1/sh.1
     260    1110    6834  /usr/man/man1/telnet.1
    2810   15643   93067  total
angel%
```

NAME

what – identify the version of files

SYNOPSIS

what files

DESCRIPTION

What searches the given *files* for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file *program.c* contains

```
char ident[] = "@(#)identification information";
```

and *program.c* is compiled to yield *program.o* and *a.out*, the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the SCCS command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

sccs(1), get(1), help(1), file(1).
The Source Code Control System in *Programming Tools for the Sun System*

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

whatis – describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

Whatis looks up a given *command* and displays the header line from the manual section. You can then run the *man*(1) command to get more information. If the line starts 'name(section) ... you can do **man section name** to get the documentation for it. Try **whatis ed** and then you should do **man 1 ed** to get the manual page for *ed*.

Whatis is actually just the **–f** option to the *man*(1) command.

FILES

/usr/lib/whatis	Data base

SEE ALSO

man(1), catman(8)

NAME

whereis – locate source, binary, and/or manual for program

SYNOPSIS

whereis [**–sbm**] [**–u**] [**–SBM** dir ... **–f**] name ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form *.ext*, for example, *.c*. Prefixes of *s.* resulting from use of source code control are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places.

OPTIONS

–b Search only for binaries.

–s Search only for sources.

–m Search only for manual sections.

–u Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus **whereis –m –u *** asks for those files in the current directory which have no documentation.

–B Change or otherwise limit the places where *whereis* searches for binaries.

–M Change or otherwise limit the places where *whereis* searches for manual sections.

–S Change or otherwise limit the places where *whereis* searches for sources.

–f Terminates the last directory list and signals the start of file names, and *must* be used when any of the **–B**, **–M**, or **–S** options are used.

EXAMPLE

Find all files in */usr/bin* which are not documented in */usr/man/man1* with source in */usr/src/cmd*:

```
angel% cd /usr/ucb
angel% whereis –u –M /usr/man/man1 –S /usr/src/cmd –f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

BUGS

Since *whereis* uses *chdir*(2) to run faster, pathnames given with the **–M**, **–S**, or **–B** must be full; that is, they must begin with a '/'.

NAME

which – locate a program file including aliases and paths (*csh* only)

SYNOPSIS

which [name] ...

DESCRIPTION

Which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's .cshrc file.

FILES

~/.cshrc source of aliases and path values

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

BUGS

Only aliases and paths from ~/.cshrc are used; importing from the current environment is not attempted. Must be executed by a csh, since only csh's know about aliases.

NAME

who – who is on the system

SYNOPSIS

who [who-file] [am I]

DESCRIPTION

Used without arguments, *who* lists the login name, terminal name, and login time for each current UNIX user. *Who* gets this information from the */etc/utmp* file.

If a filename argument is given, the named file is examined instead of */etc/utmp*. Typically the named file is */usr/adm/wtmp*, which contains a record of all logins since it was created. In this case, *who* lists logins, logouts, and crashes. Each login is listed with user name, terminal name (with '*/dev/*' suppressed), and date and time. Logouts produce a similar line without a user name. Reboots produce a line with '~' in place of the device name, and a fossil time indicating when the system went down. Finally, the adjacent pair of entries '|' and '}' indicate the system-maintained time just before and after a *date* command changed the system's idea of the time.

With two arguments, as in 'who am i' (and also 'who are you'), *who* tells who you are logged in as: it displays your hostname, login name, terminal name, and login time.

EXAMPLES

```
angel% who am I
angel!henry      ttyp0   Apr 27 11:24
angel%

krypton% who
mktg     ttym0 Apr 27 11:11
shannon  ttyp0  Apr 27 11:25
henry    ttyp1  Apr 27 11:30
krypton%
```

FILES

/etc/utmp

SEE ALSO

whoami(1), getuid(2), utmp(5), w(1)

NAME

whoami – display effective current username

SYNOPSIS

whoami

DESCRIPTION

Whoami **displays the username of whoever is currently logged in.** ***Whoami*** **works even if you are logged in as the super-user, while 'who am i' does not since it gets its information from the** */etc/utmp* **file.**

FILES

/etc/passwd username data base

SEE ALSO

who(1)

NAME

write – write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from your standard input to *user*'s screen.

When you type a *write* command, the person you're writing to sees a message like this:

Message from hostname!yourname on yourttyname at hh:mm . . .

After typing the *write* command, enter the text of your message. What you type appears line-by-line on the other user's screen. Conclude by typing an end of file indication (^D) or an interrupt. At this point *write* displays 'EOT' on your recipient's screen and exits.

To write to a user who is logged in more than once, use the *ttyname* argument to indicate the appropriate terminal name.

You can grant or deny other users permission to write to you by using the *mesg* command (default allows writing). Certain commands, *nroff* and *pr*(1) in particular, don't allow anyone to write to you while you are using them in order to prevent messy output.

If *write* finds the character '!' at the beginning of a line, it calls the shell to execute the rest of the line as a command.

Two people can carry on a conversation by *write*'ing to each other. When the other person receives the message indicating you are writing to him, he can then *write* back to you if he wishes. However, since you are now simultaneously typing and receiving messages, you end up with garbage on your screen unless you work out some sort of scheduling scheme with your partner. You might try the following conventional protocol: when you first write to another user, wait for him to write back before starting to send. Each person should end each message with a distinctive signal — **-o-** (for 'over') is standard — so that the other knows when to begin a reply. To end your conversation, type **-oo-** (for 'over and out') before finishing the conversation.

EXAMPLE

Here is an example of a short dialog between two people on different terminals. Two users called Horace and Eudora are logged in on a system called jones. To illustrate the process, both users' screens are shown side-by-side:

Eudora's Terminal	Horace's Terminal
	Horace is staring at his screen
jones% **write horace**	Message from jones!eudora on tty09 at 17:05 ...
how about a squash game tonight? -o-	how about a squash game tonight? -o-
	jones% **write eudora**
	I'm playing tiddlywinks with Carmeline -o-
Message from jones!horace on tty03 at 17:06 ...	
I'm playing tiddlywinks with Carmeline -o-	
How about the beach on Sunday? -o-	How about the beach on Sunday? -o-
	Sorry, I'm washing my tent that day -o-
Sorry, I'm washing my tent that day -o-	
See you when I get back from Peru -oo-	See you when I get back from Peru -oo-
^D	
jones%	EOF
	I hear rack of llama is very tasty -oo-
	^D
I hear rack of llama is very tasty -oo-	
EOF	jones%

FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

SEE ALSO

mesg(1), who(1), mail(1), talk(1)

NAME

xsend, xget, enroll – secret mail

SYNOPSIS

xsend *username*
xget
enroll

DESCRIPTION

These commands implement a secure communication channel; the channel is like *mail*(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use *enroll*; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use *xget*. It asks for your password, then gives you the messages.

To send secret mail, use *xsend* in the same manner as the ordinary mail command. Unlike *mail*, *xsend* accepts only one target. A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

/usr/spool/secretmail/*.key	keys
/usr/spool/secretmail/*.[0-9]	messages

SEE ALSO

mail (1)

BUGS

The knapsack public-key cryptosystem is known to be breakable.

RESTRICTIONS

These facilities are not available on software shipped outside the U.S.

NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**–v**] [**–c**] [**–**] [file]

DESCRIPTION

Xstr maintains a file called *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

xstr –c name

extracts the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

xstr

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

xstr name

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '–' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc –E name.c | xstr –c –
cc –c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

OPTIONS

–c *file* Take C source text from *file*.

–v Verbose: display a progress report indicating where new or duplicate strings were found.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' doesn't touch *strings*

SEE ALSO

mkstr(1)

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [**–vd**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

OPTIONS

–v Prepare the file *y.output* containing a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

–d Generate the file *y.tab.h* with the *define* statements that associate the *yacc*-assigned 'token codes' with the user-declared 'token names'. This allows source files other than *y.tab.c* to access the token codes.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
Yacc — Yet Another Compiler Compiler in
Programming Tools for the Sun Workstation.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

NAME

yes – be repetitively affirmative

SYNOPSIS

yes [expletive]

DESCRIPTION

Yes repeatedly outputs "y", or if *expletive* is given, that is output repeatedly. Termination is by typing an interrupt character.

NAME

rastrepl – magnify a raster image by 2 times

SYNOPSIS

rastrepl

DESCRIPTION

Rastrepl reads a file in rasterfile format (see */usr/include/rasterfile.h*) on the standard standard input, replicates each bit in both the x and y directions, and writes the resulting rasterfile to standard output.

EXAMPLES

```
tutorial% screendump | rastrepl | lpr -Pversatec -v
```

sends a rasterfile containing the current frame buffer to the lineprinter, doubling the size of the image so that it fills a single page.

SEE ALSO

screendump(1), screenload(1), lpr(1)

NAME

screendump – dump frame buffer image

SYNOPSIS

screendump [**–c**] [**–f** *display*]

DESCRIPTION

Screendump reads out the contents of the console frame buffer (*/dev/fb*) on any model of Sun Workstation and outputs the display image in Sun standard rasterfile format (see */usr/include/rasterfile.h*) on the standard output.

By default, *screendump* attempts first to output the contents of the console frame buffer. If no console frame buffer is found, *screendump* attempts to output the contents of a color frame buffer. The **–c** option selects a color frame buffer directly. Heuristics are applied to locate the color frame buffer the user is most likely interested in. The **–f** option explicitly sets the name of the desired frame buffer device.

The utility program *screenload* displays Sun standard rasterfiles on an appropriate Sun Workstation monitor. Output filters exist for printing standard monochrome rasterfiles on Versatec V-80 electrostatic plotters.

OPTIONS

–c Dump a color frame buffer without trying the console frame buffer.

–f *display*
Use *display* as the device name of the display.

EXAMPLES

```
tutorial% screendump >save.this.image
```

writes the current contents of the console frame buffer into the file *save.this.image*.

```
tutorial% screendump -f /dev/cgone0 >save.color.image
```

writes the current contents of the frame buffer */dev/cgone0* into the file *save.color.image*. This has the same effect as the **–c** option for the most common display configuration.

```
tutorial% screendump | lpr -Pversatec -v
```

sends a rasterfile containing the current frame buffer to the lineprinter, selecting the printer named "versatec" and the "v" output filter (see */etc/printcap*).

FILES

/dev/fb	Default name of console display frame buffer.
/dev/cgone0	Default name of the Sun-1 color display frame buffer.

SEE ALSO

screenload(1), lpr(1)

NAME

screenload – restore frame buffer image

SYNOPSIS

screenload [**–d**] [**–f** *display*] [**–l***index*] [**–b**] [**–g**] [**–w**] [**–h#** [**####** [...]]] [file]

DESCRIPTION

Screenload accepts input in Sun standard rasterfile format (see */usr/include/rasterfile.h*) and attempts to display the input on the appropriate monitor (monochrome or color). *Screenload* normally displays rasterfiles on the console display, but displays them on some heuristically determined color display if it finds no console display. *Screenload* displays color rasterfiles only on a color monitor. *Screenload* is able to display monochrome rasters on a color display.

If the dimensions of the image contained in the input data are smaller than the actual resolution of the display (e.g. loading a 1024-by-800 Sun-1 image on a 1152-by-900 Sun-2 display), *screenload* centers the image on the actual workstation screen and fills the border area with solid black (by default). Various options may be used to change the fill pattern.

If the dimensions of the image contained in the input data are larger than the actual resolution of the display, *screenload* clips the right and bottom edges of the input image.

If there is an optional filename argument, *screenload* reads its input data from that file. If no filename argument is given, *screenload* reads its input data from the standard input.

The utility program *screendump* creates Sun standard rasterfiles from the display on a Sun Workstation monitor.

OPTIONS

–d Verify that display dimensions and input data image dimensions match, and print warning messages if they do not.

–f *display*
Use *display* as the name of the frame buffer device.

–l*index* If the image is smaller than the display, use *index* (0 <= *index* < 256) as the index value into the color map for the foreground color of the border fill pattern. The default index value is 255.

–b If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of solid ones (*default*). On a monochrome display this results in a black border; on a color display the color map value selected by the –l option determines the border color.

–g If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of "desktop grey". On a monochrome display this results in a border matching the background pattern used by the SunWindows system; on a color display the color map value selected by the –l option determines the foreground border color, though the pattern is the same as on a monochrome display.

–w If the input data dimensions are smaller than the display dimensions, fill the border with a pattern of solid zeros. On a monochrome display this results in a white border; on a color display the color map value at index 0 determines the border color.

–h# If the input data dimensions are smaller than the display dimensions, fill the border with the bit pattern described by the following # 16-bit hexadecimal constants. Note that a "1" bit is black and a "0" bit is white on the monochrome display; on a color diplay the color map value selected by the "– i" option determines the border foreground color. The number of hex constants in the pattern is limited to 16.

EXAMPLES

tutorial% screenload saved.display.image

reloads the raster image contained in the file *saved.display.image* on the display type indicated by

the rasterfile header in that file.

```
tutorial% screenload -f/dev/cgone0 monochrome.image
```

reloads the raster image in the file *monochrome.image* on the frame buffer device */dev/cgone0*.

```
tutorial% screenload -h1 ffff sun_1.saved.image
```

is equivalent to the **-b** option (fill border with black), while

```
tutorial% screenload -h4 8888 8888 2222 2222 sun_1.saved.image
```

is equivalent to the **-g** option (fill border with desktop grey).

FILES

/dev/fb	Default name of console display frame buffer
/dev/cgone0	Default name of SUn-1 color display frame buffer

SEE ALSO

screendump(1)

NAME

vfontinfo – inspect and print out information about UNIX fonts

SYNOPSIS

vfontinfo [–v] fontname [characters]

DESCRIPTION

Vfontinfo displays information about fonts in the UNIX format. *Vfontinfo* displays all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format. *Vfontinfo* displays the information on the standard output.

The optional arguments are as follows:

fontname is the name of the font you wish to inspect. If it can't find the file in your working directory, *vfontinfo* looks in */usr/lib/vfont* (the place most of the fonts are kept).

characters specify certain characters to show. If omitted, the entire font is shown.

OPTIONS

–v display the bits of the glyph itself as an array of X's and spaces, in addition to the header information.

SEE ALSO

vfont(5)

NAME

vtroff – troff to a raster plotter

SYNOPSIS

vtroff [**–w**] [**–F** *majorfont*] [**–123** *minorfont*] [**–l***length*] [**–x**] troff arguments

DESCRIPTION

Vtroff runs *troff*(1) sending its output through various programs to produce typeset output on a raster plotter such as a Benson-Varian or or a Versatec.

OPTIONS

–W use a wide output device; the default is to use a narrow device.

–l*length*

(lower case ell) split the output onto successive pages every *length* inches rather than the default 11 inches.

–F *majorfont*

specify *majorfont* as the font to use instead of the default Hershey font. This places normal, italic and bold versions of *majorfont* on font positions 1, 2, and 3.

–n *minorfont*

place a font specified by *minorfont* only on a single position specified by **n**, where **n** is 1, 2, or 3. *Vtroff* adds a *.r* to the minor font name if needed. Thus

```
tutorial% vtroff –ms paper
```

will set a paper in the Hershey font, while

```
tutorial% vtroff –F nonie –ms paper
```

will set the paper in the (sans serif) nonie font.

–x exactly simulate photo-typesetter output — that is, using the width tables for the C/A/T photo-typesetter.

FILES

/usr/lib/tmac/tmac.vcat	default font mounts and bug fixes
/usr/lib/fontinfo/*	fixes for other fonts
/usr/lib/vfont	directory containing fonts

SEE ALSO

troff(1), vfont(5), vpr(1)

BUGS

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Versatec fonts have different widths for individual characters than the fonts found on the typesetter, the following dodge is necessary: If you don't use the *troff* **.fp** directive, you get the widths of the standard typesetter fonts for the C/A/T. If, however, you remount the R, I, B and S fonts, you get the width tables for the Versatec.

NAME

vwidth – make troff width table for a font

SYNOPSIS

vwidth fontfile pointsize > ft*xx*.c
cc -c ft*xx*.c
mv ft*xx*.o /usr/lib/font/ft*xx*

DESCRIPTION

Vwidth translates from the width information stored in the *vfont* style format to the format expected by *troff*. *Troff* wants an object file in *a.out*(5) format — this fact does not seem to be documented anywhere in the *troff* manuals. *Troff* should look directly in the font file but it doesn't.

It is not necessary to use *vwidth* unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use *vwidth* if the physical width of the glyph (that is, the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and *vwidth* run.

Vwidth produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the *.o* file) saved. The resulting file should be placed in */usr/lib/font* in the file ft*xx* where *xx* is a one or two letter code that is the logical (internal to troff) font name. This name can be found by looking in the file /usr/lib/fontinfo/*fname** where *fname* is the external name of the font.

SEE ALSO

vfont(5), troff(1),

BUGS

Produces the C file using obsolete syntax that the portable C compiler complains about.

NAME

adventure – an exploration game

SYNOPSIS

/usr/games/adventure

DESCRIPTION

The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type 'quit'; to save a game for later resumption, type 'suspend'.

BUGS

Saving a game creates a large executable file instead of just the information needed to resume the game.

NAME

arithmetic – provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [+−x/] [range]

DESCRIPTION

Arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+−x/** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is **+−**

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon – the game of backgammon

SYNOPSIS

backgammon [–] [n r w b pr pw pb t*term* s*file*]

DESCRIPTION

Backgammon lets you play backgammon against the computer or against a 'friend'. All commands only are one letter, so you don't need to type a carriage return, except at the end of a move. *Backgammon* is mostly self documenting, so that a question mark (?) will usually get some help. If you answer 'y' when *backgammon* asks if you want the rules, you will get text explaining the rules of the game, some hints on strategy, instruction on how to use *backgammon,* and a tutorial consisting of a practice game against the computer. A description of how to use *backgammon* can be obtained by answering 'y' when it asks if you want instructions. The possible arguments for backgammon (most are unnecesary but some are very convenient) consist of:

n don't ask for rules or instructions

r player is red (implies n)

w player is white (implies n)

b two players, red and white (implies n)

pr print the board before red's turn

pw print the board before white's turn

pb print the board before both player's turn

t*term* terminal is type *term*, uses */etc/termcap*, otherwise uses the TERM environment variable.

s*file* recover previously saved game from *file*. This can also be done by executing the saved file, that is, typing its name in as a command.

Arguments may be optionally preceded by a – sign. Several arguments may be concatenated together, but not after 's' or 't' arguments, since they can be followed by an arbitrary string. Any unrecognized arguments are ignored. An argument of a lone – gets a description of possible arguments.

If *term* has capabilities for direct cursor movement. *backgammon* 'fixes' the board after each move, so the board does not need to be reprinted, unless the screen suffers some horrendous malady. Also, any 'p' option will be ignored.

QUICK REFERENCE

When *backgammon* prompts by typing only your color, type a space or carriage return to roll, or

d to double

p to print the board

q to quit

s to save the game for later

When *backgammon* prompts with 'Move:', type

p to print the board

q to quit

s to save the game

or a *move,* which is a sequence of

s-f move from **s** to **f**

s/r move one man on **s** the roll **r** separated by commas or spaces and ending with a newline. Available abbreviations are

s-f1-f2 means s-f1,f1-f2

s/r1r2 means s/r1,s/r2

Use 'b' for bar and 'h' for home, or 0 or 25 as appropriate.

FILES

/usr/games/teachgammon rules and tutorial
/etc/termcap terminal capabilities

BUGS

***Backgammon's* strategy needs much work.**

NAME

banner – print large banner on printer

SYNOPSIS

/usr/games/banner [**–w***n*] message ...

DESCRIPTION

Banner prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If **–w** is given, the output is scrunched down from a width of 132 to *n* , suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS

Several ASCII characters are not defined, notably <, >, [,], \, ^, _ {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The **–w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

NAME

bcd – convert to antique media

SYNOPSIS

/usr/games/bcd text

DESCRIPTION

Bcd **converts the literal** *text* **into a form familiar to old-timers.**

SEE ALSO

dd(1)

NAME

bdemos – demonstrate Sun Monochrome Bitmap Display

SYNOPSIS

/usr/demo/bballs
/usr/demo/bbounce
/usr/demo/bdemos
/usr/demo/bjump
/usr/demo/bphoto *file*
/usr/demo/brotcube

DESCRIPTION

Bdemos is a collection of simple demonstration programs for the Sun Monochrome Bitmap Display. Each program is briefly described below. Unless otherwise noted, each program should be terminated by typing the appropriate key (usually DELETE or ^C) to generate an interrupt signal.

bballs colliding balls demo

bbounce bouncing square demo

bdemos a collection of demos

This program has a menu for selection of several different demos. After typing a key to select a particular demo, the user may type ^C to get back the menu. Type 'q' to quit.

bjump simulated jump to hyperspace

bphoto *file*
dither monochrome image *file* to bitmap display

Image files suitable for display by this program are in */usr/demo/bwpix*.

brotcube black and white spinning cube

FILES

/usr/demo/bwpix

SEE ALSO

draw(6)

NAME

boggle – play the game of boggle

SYNOPSIS

/usr/games/boggle [+] [++]

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl eple moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 + 's as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

bsuncube – view 3-D Sun logo

SYNOPSIS

/usr/demo/bsuncube

DESCRIPTION

Bsuncube allows the user to view a cube from various positions with hidden faces removed. The faces of the cube consist of the Sun logo. The viewing position is selected using the mouse. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display.

The program operates in two modes: DisplayObject mode and SelectView mode. The program starts in DisplayObject mode.

DisplayObject: The cube is displayed in 3-D perspective with hidden faces removed. Type "q" while in this mode to exit the program. Hit mouse button 3 to switch to SelectView mode.

SelectView: Schematic projections of the outline of the cube are shown and the mouse is used to select a viewing position. Use button 1 to set x and button 2 to set y in the *Front View*. Use button 2 to set z in the *Top View*. Hit button 3 to switch to DisplayObject mode.

The view shown in DisplayObject mode is drawn using the conventions that the viewer is always looking from the viewing position toward the center of the cube and that the positive y axis on the screen is the projection of the positive y axis in 3-D cube coordinates.

NAME

canfield, cfscores – the solitaire card game canfield

SYNOPSIS

/usr/games/canfield
/usr/games/cfscores

DESCRIPTION

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase ones chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (the initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

FILES

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

NAME

chase – Try to escape to killer robots

SYNOPSIS

/usr/games/chase [*nrobots*] [*nfences*]

DESCRIPTION

The object of the game chase is to move around inside of the box on the screen without getting eaten by the robots chasing and without running into anything.

If a robot runs into another robot while chasing you, they crash and leave a junk heap. If a robot runs into a fence, it is destroyed.

If you can survive until all the robots are destroyed, you have won!

If you do not specify either *nrobots* or *nfences*, chase will prompt you for them.

NAME

ching – the book of changes and other cookies

SYNOPSIS

/usr/games/ching [hexagram]

DESCRIPTION

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams*, each symbolized by a particular arrangement of six straight (——) and broken (– –) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each hexagram consists of two major sections. The **Judgement** relates specifically to the matter at hand (E.g., "It furthers one to have somewhere to go.") while the **Image** describes the general attributes of the hexagram and how they apply to one's own life ("Thus the superior man makes himself strong and untiring.").

When any of the lines has the value six or nine, it is a moving line; for any such line there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second hexagram (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow–stalks or tossed coins. The resulting hexagram will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, the UNIX *oracle* simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process id and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin–toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try *fortune*(6).

SEE ALSO

It furthers one to see the great man.

DIAGNOSTICS

The great prince issues commands,
Founds states, vests families with fiefs.
Inferior people should not be employed.

BUGS

Waiting in the mud
Brings about the arrival of the enemy.

If one is not extremely careful,
Somebody may come up from behind and strike him.
Misfortune.

NAME

colordemos – demonstrate Sun Color Graphics Display

SYNOPSIS

/usr/demo/cballs
/usr/demo/cdraw
/usr/demo/cphoto *file*
/usr/demo/cpipes
/usr/demo/cshowmap *file*
/usr/demo/csnow
/usr/demo/csuncube
/usr/demo/csunlogo
/usr/demo/cvlsi

DESCRIPTION

Colordemos is a collection of simple demonstration programs for the Sun Color Graphics Display. Each program is briefly described below. To exit each program, send an interrupt signal by typing the appropriate key (usually CONTROL C).

cballs colliding balls on color display

cdraw draw on the color display; see *draw*(6) for an explanation of how to use *cdraw*.

cphoto *file*
display dithered color file on color display. Files suitable for display are in */usr/demo/colorpix*.

cpipes colliding pipes on color display

cshowmap *file*
display maps. Files suitable for display are in */usr/demo/segments*.

csnow color kaleidoscope

csuncube multicolored Sun logo

csunlogo shaded Sun logo

cvlsi color vlsi layout demo

FILES

/usr/demo/colorpix
/usr/demo/segments

NAME

cribbage – the card game cribbage

SYNOPSIS

/usr/games/cribbage [**–req**] *name ...*

DESCRIPTION

Cribbage plays the card game cribbage, with *cribbage* playing one hand and the user the other. *Cribbage* initially asks the user if the rules of the game are needed – if so, *cribbage* displays the appropriate section from *According to Hoyle* with *more*(1).

OPTIONS

–e Provide an explanation of the correct score when the player makes mistakes scoring his hand or crib. This is especially useful for beginning players.

–q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

–r Instead of asking the player to cut the deck, *cribbage* will randomly cut the deck.

PLAYING CRIBBAGE

Cribbage first asks the player whether he wishes to play a short game ("once around", to 61) or a long game ("twice around", to 121). A response of 's' results in a short game, any other response plays a long game.

At the start of the first game, *cribbage* asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, *cribbage* first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, *cribbage* cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the later case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. *Cribbage* keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. *Cribbage* requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

SPECIFYING CARDS

Cards are specified as *rank* followed by *suit*. The *ranks* may be specified as one of **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **t**, **j**, **q**, and **k**, or alternatively, one of **ace**, **two**, **three**, **four**, **five**, **six**, **seven**, **eight**, **nine**, **ten**, **jack**, **queen**, and **king**. *Suits* may be specified as **s**, **h**, **d**, and **c**, or alternatively as **spades**, **hearts**, **diamonds**, and **clubs**. A card may be specified as *rank suit*, or *rank* of *suit*. If the single letter *rank* and *suit* designations are used, the space separating the *suit* and *rank* may be left out. Also, if only one card of the desired *rank* is playable, typing the *rank* is sufficient. For example, if your hand was **2h**, **4d**, **5c**, **6h**, **jc**, **kd** and you wanted to discard the king of diamonds, you could type any of **k**, **king**, **kd**, **k d**, **k of d**, **king d**, **king of d**, **k diamonds**, **k of diamonds**, **king diamonds**, or **king of diamonds**.

FILES

/usr/games/cribbage

NAME

draw – interactive graphics drawing

SYNOPSIS

/usr/demo/bdraw
/usr/demo/cdraw

DESCRIPTION

The *draw* programs are menu-driven programs which use the mouse, keyboard, bitmap display and optionally the color display to draw objects, drag them around, save them on disk, and so on. *Bdraw* is the draw program for the black and white display and *cdraw* is the program for driving the color display.

The main menu items are selected by moving the mouse cursor and pressing the left mouse button. To redraw the display, point at the left edge of the main menu box and press the left button. The main menu items are:

New Seg xlate
Open a new translatable segment. A segment is a collection of attributes and primitives (lines, text, polygons, etc.). A translatable segment may subsequently be positioned.

New Seg xform
Open a new transformable segment. A transformable segment may subsequently be rotated, scaled, or positioned.

Delete Seg
To delete a segment, point at any primitive in the segment and press the left button.

Lines To add line primitives to the currently open segment, position cursor, press the left button, ... press right button to quit.

Polygon To add a polygon primitive to the currently open segment, position the cursor, press the left button, ... press the right button to terminate the boundary definition. Polygons are filled with the current fill attribute.

Raster To add a raster primitive to the currently open segment, position the cursor, press the left button to reposition the box, adjust the box by moving the mouse, press the right button to create the raster primitive comprising the boxed bitmap. A 'rasterfile' is also created on disk for hardcopy purposes (see */usr/include/rasterfile.h*). This 'rasterfile' file may be spooled to a Versatec printer/plotter for hardcopy after exiting from the draw program. The command to do this is **lpr –v rasterfile**.

Text To add a text primitive to the currently open segment, position cursor, press left button, type the text string at the keyboard (back space works), hit return. Text is drawn with the current text attributes.

Marker To add marker primitives to the currently open segment, position cursor, press the left button to place marker, ... press the right button to quit.

Position To position a segment, point at any primitive in the segment, press left button, position the segment, press right button to quit.

Rotate To rotate a transformable segment, point at any primitive in the segment, press left button, move mouse to rotate, press right button to quit.

Scale To scale a transformable segment, point at any primitive in the segment, press the left button, move mouse to scale in x or y, press right button to quit.

Attributes
This item brings up the attribute menu. To select an attribute such as text font, region fill texture (color), linestyle, or line width, point at the item and press the left button. Point at the left edge of the menu box to quit.

Save Seg To save a segment on a disk file, point at the segment, press the left button, type the disk file name, hit return.

Restore Seg
To restore a previously saved segment from disk, type file name, hit return.

Exit Exit the draw program.

BUGS

Rasters and raster text do not scale or rotate. If segments completely overlap, only the last one drawn may be picked by pointing with the mouse. This also applies to the menu segments! Therefore, don't cover them up with polygons. If the draw program is interrupted (del key) the 'reset' command must be given to turn keyboard echo back on and to reset -cbreak. Therefore, use the Exit item in the main menu to exit the program.

NAME

fish – play "Go Fish"

SYNOPSIS

/usr/games/fish

DESCRIPTION

Fish plays the game of "Go Fish", a children's card game. The object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; the default is pretty dumb.

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [–] [–wslao]

DESCRIPTION

Fortune with no arguments prints out a random adage. The flags mean:

–w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

–s Short messages only.

–l Long messages only.

–o Choose from an alternate list of adages, often used for potentially offensive ones.

–a Choose from either list of adages.

FILES

/usr/games/lib/fortunes.dat

NAME

hangman – Computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In *hangman,* the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words On-line word list

NAME

mille – play Mille Bornes

SYNOPSIS

/usr/games/mille [file]

DESCRIPTION

Mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bornes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

- **P** Pick a card from the deck. This card is placed in the 'P' slot in your hand.
- **D** Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a carriage-return or space. The carriage-return or space is required to allow recovery from typos which can be very expensive, like discarding safeties.
- **U** Use a card. The card is again indicated by its number, followed by a carriage-return or space.
- **O** Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.
- **Q** Quit the game. This will ask for confirmation, just to be sure. Hitting <DELETE> (or <RUBOUT>) is equivalent.
- **S** Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a carriage-return without a name, the save will be terminated and the game resumed.
- **R** Redraw the screen from scratch. The command ^L (control 'L') will also work.
- **W** Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save. (The game itself is a product of Parker Brothers, Inc.)

SEE ALSO

curses(3X)

CARDS

Here is some useful information. The number in brackets after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas [2]	Gasoline [6]	Extra Tank [1]
Flat Tire [2]	Spare Tire [6]	Puncture Proof [1]
Accident [2]	Repairs [6]	Driving Ace [1]
Stop [4]	Go [14]	Right of Way [1]
Speed Limit [3]	End of Limit [6]	

25 – [10], 50 – [10], 75 – [10], 100 – [12], 200 – [4]

RULES

Object: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. With the exception of the *speed limit* card, they can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA** (unlabeled): This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaches 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastyness.

Go (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

Stop is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

Speed Limit is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

End of Limit is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

Out of Gas is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

Flat Tire is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and

then a *Go* card before they can play any more mileage.

Accident is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn.*

Right of Way prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.

Extra Tank When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

Puncture Proof When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

Driving Ace When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand.* A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action.*

Coup Fourć: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bornes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.

Each Safety: 100 points for each safety in the Safety area.

All 4 Safeties: 300 points if all four safeties are played.

Each Coup Fourć: 300 points for each Coup Fourć accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.

Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.

Delayed Action: 300 points bonus for finishing after the deck was exhausted.

Extension: 200 points bonus for completing a 1000 mile trip.

Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop – Monopoly game

SYNOPSIS

/usr/games/monop [file]

DESCRIPTION

Monop is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", *i.e.*, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, e.g., a name, place or person, you can type '?' to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

A Summary of Commands:

quit: quit game: This allows you to quit the game. It asks you if you're sure.

print: print board: This prints out the current board. The columns have the following meanings (column headings are the same for the **where, own holdings, and holdings** commands):

- Name — The first ten characters of the name of the square
- Own — The *number* of the owner of the property.
- Price — The cost of the property (if any)
- Mg — This field has a '*' in it if the property is mortgaged
- # — If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
- Rent — Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings: List your own holdings, *i.e.*, money, get-out-of-jail-free cards, and property.

holdings: holdings list: Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type "done".

shell: shell escape: Escape to a shell. When the shell dies, the program continues where you left off.

mortgage: mortgage property: Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage: unmortgage property: Unmortgage mortgaged property.

buy: buy houses: Sets up a list of monopolies on which you can buy houses. If there is

more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: sell houses: Sets up a list of monopolies from which you can sell houses. it operates in an analogous manner to *buy*

card: card for jail: Use a get-out-of-jail-free card to get out of jail. If you're not in jail, or you don't have one, it tells you so.

pay: pay for jail: Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you're not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: save game: Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the *monop* command, or by using the *restore* command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore: restore game: Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the <RETURN> key instead of a command, it is the same as typing *roll*.

FILES

/usr/games/lib/cards.pck Chance and Community Chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

number – convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

Number **copies the standard input to the standard output, changing each decimal number to a fully spelled out version.**

NAME

quiz – test your knowledge

SYNOPSIS

/usr/games/quiz [**–i** file] [**–t**] [category1 category2]

DESCRIPTION

Quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, *quiz* gives instructions and lists the available categories.

Quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **–t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **–i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category ':' line
category  = alternate | category '|' alternate
alternate = empty | alternate primary
primary   = character | '[' category ']' | option
option    = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with *sh*(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain – animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

Rain's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use *termcap*, the TERM environment variable must be set (and exported) to the type of the terminal being used.

FILES

/etc/termcap

NAME

sail – multi-user wooden ships and iron men

SYNOPSIS

sail [–x] [num]

DESCRIPTION

Sail is a computer version of Avalon Hill's game of fighting sail originally developed by S. Craig Taylor.

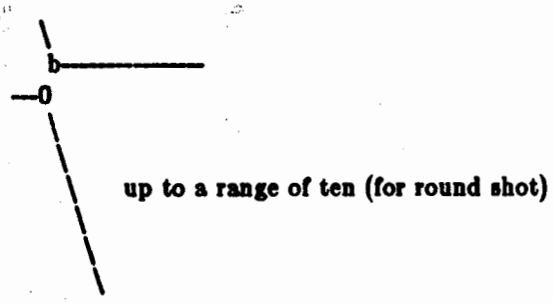
NOTES

Sail is really two programs in one. Each player keeps track of his own ship plus a *DRIVER* program is execl'd (by the first player) to keep track of the computer's ships.

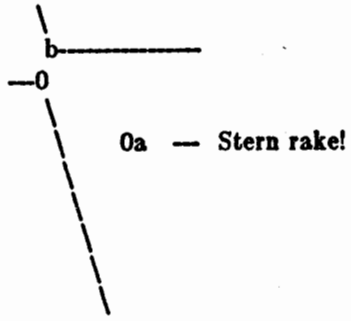
The player is given the first available ship in a scenario and the computer takes the rest. Obviously the more ships in your game, the longer the *DRIVER* will take to move them. If additional players join the game, they will be given ships and the *DRIVER* will have less work to do.

HISTORICAL INFO

Old Square Rigger's were very maneuverable ships capable of intricate sailing. Their one main disadvantage was being unable to sail very close to the wind. The design of wooden ship allowed only for the guns to bear to the left and right sides. A few guns of small aspect (usually 6 or 9 pounders) could point forward, but their effect would be small compared to a 68 gun broadside of 24 pounders. The guns bear approximately like so:



An interesting phenomenon occurred when a broadside could fire down the length of an enemy ship. The shot tended to bounce along the deck and did several times more damage. This phenomenon was called a rake. It happened that a stern rake (firing from the stern to the bow) occasioned more damage than a bow rake, so that was the most desirable.



Most ships were equipped with Carronades which were very large, close range cannons. The carronades have a range of two in this game and can considerably add to your fire-power when they come to bear. If the distance to the target ship is greater than 6, the guns can only fire at the rigging. A ship's guns could fire a variety of ammunition. For example:

ROUND

Range of 10. Good for hull or rigging hits.

DOUBLE

Range of 1. Extra good for hull or rigging hits. Double takes two turns to load.

CHAIN

Range of 3. Excellent for tearing down rigging. Cannot damage hull or guns, though.

GRAPE

Range of 1. Devastating against enemy crews.

When a ship has been battered into a hulk (zero hull), it has no choice but to surrender to the firing ship. This ceremony is called 'striking your colours.' A struck ship has a chance of exploding or sinking after a while. When a ship surrenders, its point value is given to the aggressor. When a ship is captured, twice the point value is awarded the victor.

Normally, ships sailed into battle with greatly shortened sail to avoid excessive damage to the precious rigging. However, in this game the player can increase to full sails and move much faster if he wishes. But, all rigging hits incurred with full sails set are doubled. The direction rose displayed on the sample screen gives the maximum speeds possible for a specific ship at all attitudes to the wind. The full sail speeds are in parenthesis.

Repairs can be made at the slow rate of two (hull, gun, or rigging) hits restored per three turns.

Ships of class 3 or greater drift when there is wind at the rate of one 'square' per turn. Ships of the Line drift one 'square' every other turn.

INSTRUCTIONS

Sail follows the Avalon Hill advanced rules very closely using the optional rules for 'exploding ships', 'full sails', and some others. A few unique commands have been added which seemed to be helpful on the reduced screen. 'i' is such a command.

Boarding had to go through a major revision. To prevent immediate capture of an unprepared crew (fouling is often not reported until after boarding has commenced) the boarded ship automatically fights defensively (at a small disadvantage) if no DBP's have been prepared.

The Order of Play has been eliminated for the player, but the *DRIVER* still abides by it.

The commands for the player were designed to be as intelligent as possible to save typing. Some of the nuances I developed should be explained.

- Your prompt

The others I will illustrate with examples.

```
move(3, 2): r1l      /* 3 movements max, of which two
                        may be 45' turns. */

move(3,'2): 1r1      /* 3 movements max of which two may
                        be 45' turns, but the ship must
                        move ahead before turning (there
                        is a loss of headway after
                        drifting) */

move(0,'0): r        /* You can always make one turn
                        even when you can't move straight
                        ahead. */
```

If you are grappled, fouled, or out of crew, you cannot move of course.

COMMANDS

'f' Fire broadsides if they bear
'l' Reload
'm' Move (see above & below)
'i' Ask lookout for closest ship
'I' Ask lookout for closest enemy ship
's' Send a message around the fleet
'b' Attempt to board an enemy ship
'L' Unload broadsides (to change ammo)
'B' Recall boarding parties
'c' Change set of sail
'r' Repair
'u' Attempt to unfoul
'g' Grapple/ungrapple
'^L' Redraw screen
'q' Quit

SCENARIOS

Ranger vs. Drake:

Wind from the N, blowing a fresh breeze.

(a) Ranger 19 gun Sloop (crack crew) (7 pts)
(b) Drake 17 gun Sloop (crack crew) (6 pts)

The Battle of Flamborough Head:

Wind from the S, blowing a fresh breeze.

This is John Paul Jones' first famous battle. Aboard the Bonhomme Richard, he was able to overcome the Serapis's greater firepower by quickly boarding her.

(a) Bonhomme Rich 42 gun Corvette (crack crew) (11 pts)
(b) Serapis 44 gun Frigate (crack crew) (12 pts)

Arbuthnot and Des Touches:

Wind from the N, blowing a gale.

(b) America 64 gun Ship of the Line (crack crew) (20 pts)
(b) Befford 74 gun Ship of the Line (crack crew) (26 pts)
(b) Adamant 50 gun Ship of the Line (crack crew) (17 pts)
(b) London 98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Royal Oak 74 gun Ship of the Line (crack crew) (26 pts)
(f) Neptune 74 gun Ship of the Line (average crew) (24 pts)
(f) Duc Bougogne 80 gun 3 Decker SOL (average crew) (27 pts)
(f) Conquerant 74 gun Ship of the Line (average crew) (24 pts)
(f) Provence 64 gun Ship of the Line (average crew) (18 pts)
(f) Romulus 44 gun Ship of the Line (average crew) (10 pts)

Suffren and Hughes:

Wind from the S, blowing a fresh breeze.

(b) Monmouth 74 gun Ship of the Line (average crew) (24 pts)
(b) Hero 74 gun Ship of the Line (crack crew) (26 pts)
(b) Isis 50 gun Ship of the Line (crack crew) (17 pts)
(b) Superb 74 gun Ship of the Line (crack crew) (27 pts)
(b) Burford 74 gun Ship of the Line (average crew) (24 pts)

(f) Flamband 50 gun Ship of the Line (average crew) (14 pts)
(f) Annibal 74 gun Ship of the Line (average crew) (24 pts)
(f) Severe 64 gun Ship of the Line (average crew) (18 pts)
(f) Brilliant 80 gun Ship of the Line (crack crew) (31 pts)
(f) Sphinx 80 gun Ship of the Line (average crew) (27 pts)

Nymphe vs. Cleopatre:
Wind from the S, blowing a fresh breeze.

(b) Nymphe 36 gun Frigate (crack crew) (11 pts)
(f) Cleopatre 36 gun Frigate (average crew) (10 pts)

Mars vs. Hercule:
Wind from the S, blowing a fresh breeze.
(b) Mars 74 gun Ship of the Line (crack crew) (26 pts)
(f) Hercule 74 gun Ship of the Line (average crew) (23 pts)

Ambuscade vs. Baionnaise:
Wind from the N, blowing a fresh breeze.

(b) Ambuscade 32 gun Frigate (average crew) (9 pts)
(f) Baionnaise 24 gun Corvette (average crew) (9 pts)

Constellation vs. Insurgent:
Wind from the S, blowing a gale.

(a) Constellation 38 gun Corvette (elite crew) (17 pts)
(f) Insurgent 36 gun Corvette (average crew) (11 pts)

Constellation vs. Vengeance:
Wind from the S, blowing a fresh breeze.

(a) Constellation 38 gun Corvette (elite crew) (17 pts)
(f) Vengeance 40 gun Frigate (average crew) (15 pts)

The Battle of Lissa:
Wind from the S, blowing a fresh breeze.

(b) Amphion 32 gun Frigate (elite crew) (13 pts)
(b) Active 38 gun Frigate (elite crew) (18 pts)
(b) Volage 22 gun Frigate (elite crew) (11 pts)
(b) Cerberus 32 gun Frigate (elite crew) (13 pts)
(f) Favorite 40 gun Frigate (average crew) (15 pts)
(f) Flore 40 gun Frigate (average crew) (15 pts)
(f) Danae 40 gun Frigate (crack crew) (17 pts)
(f) Bellona 32 gun Frigate (green crew) (9 pts)
(f) Corona 40 gun Frigate (green crew) (12 pts)
(f) Carolina 32 gun Frigate (green crew) (7 pts)

Constitution vs. Guerriere:
Wind from the SW, blowing a gale.

(a) Constitution 44 gun Corvette (elite crew) (24 pts)
(b) Guerriere 38 gun Frigate (crack crew) (15 pts)

United States vs. Macedonian:
Wind from the S, blowing a fresh breeze.

(a) United States 44 gun Frigate (elite crew) (24 pts)
(b) Macedonian 38 gun Frigate (crack crew) (16 pts)

Constitution vs. Java:
Wind from the S, blowing a fresh breeze.

(a) Constitution 44 gun Corvette (elite crew) (24 pts)
(b) Java 38 gun Corvette (crack crew) (19 pts)

Chesapeake vs. Shannon:
Wind from the S, blowing a fresh breeze.

(a) Chesapeake 38 gun Frigate (average crew) (14 pts)
(b) Shannon 38 gun Frigate (elite crew) (17 pts)

The Battle of Lake Erie:
Wind from the S, blowing a light breeze.

(a) Lawrence 20 gun Sloop (crack crew) (9 pts)
(a) Niagara 20 gun Sloop (elite crew) (12 pts)
(b) Lady Prevost 13 gun Brig (crack crew) (5 pts)
(b) Detroit 19 gun Sloop (crack crew) (7 pts)
(b) Q. Charlotte 17 gun Sloop (crack crew) (6 pts)

Wasp vs. Reindeer:
Wind from the S, blowing a light breeze.

(a) Wasp 20 gun Sloop (elite crew) (12 pts)
(b) Reindeer 18 gun Sloop (elite crew) (9 pts)

Constitution vs. Cyane and Levant:
Wind from the S, blowing a moderate breeze.

(a) Constitution 44 gun Corvette (elite crew) (24 pts)
(b) Cyane 24 gun Sloop (crack crew) (11 pts)
(b) Levant 20 gun Sloop (crack crew) (10 pts)

Pellew vs. Droits de L'Homme:
Wind from the N, blowing a gale.

(b) Indefatigable 44 gun Frigate (elite crew) (14 pts)
(b) Amazon 36 gun Frigate (crack crew) (14 pts)
(f) Droits L'Hom 74 gun Ship of the Line (average crew) (24 pts)

Algeciras:
Wind from the SW, blowing a moderate breeze.

(b) Caesar 80 gun Ship of the Line (crack crew) (31 pts)
(b) Pompee 74 gun Ship of the Line (crack crew) (27 pts)
(b) Spencer 74 gun Ship of the Line (crack crew) (26 pts)
(b) Hannibal 98 gun 3 Decker SOL (crack crew) (28 pts)
(s) Real-Carlos 112 gun 3 Decker SOL (green crew) (27 pts)
(s) San Fernando 96 gun 3 Decker SOL (green crew) (24 pts)
(s) Argonauta 80 gun Ship of the Line (green crew) (23 pts)
(s) San Augustine 74 gun Ship of the Line (green crew) (20 pts)
(f) Indomptable 80 gun Ship of the Line (average crew) (27 pts)
(f) Desaix 74 gun Ship of the Line (average crew) (24 pts)

Lake Champlain:
Wind from the N, blowing a fresh breeze.

(a) Saratoga	26 gun Sloop (crack crew) (12 pts)
(a) Eagle	20 gun Sloop (crack crew) (11 pts)
(a) Ticonderoga	17 gun Sloop (crack crew) (9 pts)
(a) Preble	7 gun Brig (crack crew) (4 pts)
(b) Confiance	37 gun Frigate (crack crew) (14 pts)
(b) Linnet	16 gun Sloop (elite crew) (10 pts)
(b) Chubb	11 gun Brig (crack crew) (5 pts)

Last Voyage of the USS President:
Wind from the N, blowing a fresh breeze.

(a) President	44 gun Frigate (elite crew) (24 pts)
(b) Endymion	40 gun Frigate (crack crew) (17 pts)
(b) Pomone	44 gun Frigate (crack crew) (20 pts)
(b) Tenedos	38 gun Frigate (crack crew) (15 pts)

Hornblower and the Natividad:
Wind from the E, blowing a gale.

A scenario for you Horny fans. Remember, he sank the Natividad against heavy odds and winds. Hint: don't try to board the Natividad, her crew is much bigger, albeit green.

(b) Lydia	36 gun Frigate (elite crew) (13 pts)
(s) Natividad	50 gun Ship of the Line (green crew) (14 pts)

Curse of the Flying Dutchman:
Wind from the S, blowing a fresh breeze.

Just for fun, take the Piece of cake.

(s) Piece of Cake	24 gun Corvette (average crew) (9 pts)
(f) Flying Dutchy	120 gun 3 Decker SOL (elite crew) (43 pts)

The South Pacific:
Wind from the S, blowing a strong breeze.

(a) USS Scurvy	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(b) HMS Tahiti	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Australian	32 gun Frigate (average crew) (9 pts)
(f) Bikini Atoll	7 gun Brig (crack crew) (4 pts)

Hornblower and the battle of Rosas
Wind from the E, blowing a fresh breeze.

The only battle Hornblower ever lost. He was able to dismast one ship and stern rake another's though. See if you can do as well.

(b) Sutherland	74 gun Ship of the Line (crack crew) (26 pts)
(f) Turenne	80 gun 3 Decker SOL (average crew) (27 pts)
(f) Nightmare	74 gun Ship of the Line (average crew) (24 pts)
(f) Paris	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Napolean	74 gun Ship of the Line (green crew) (20 pts)

Cape Horn:
Wind from the NE, blowing a strong breeze.

(a) Concord	80 gun Ship of the Line (average crew) (27 pts)
(a) Berkeley	98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Thames	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Madrid	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Musket	80 gun 3 Decker SOL (average crew) (27 pts)

New Orleans:
Wind from the SE, blowing a fresh breeze.

Watch that little Cypress go!

(a) Alligator	120 gun 3 Decker SOL (elite crew) (43 pts)
(b) Firefly	74 gun Ship of the Line (crack crew) (27 pts)
(b) Cypress	44 gun Frigate (elite crew) (14 pts)

Botany Bay:
Wind from the N, blowing a fresh breeze.

(b) Shark	64 gun Ship of the Line (average crew) (18 pts)
(f) Coral Snake	44 gun Corvette (elite crew) (24 pts)
(f) Sea Lion	44 gun Frigate (elite crew) (24 pts)

Voyage to the Bottom of the
Wind from the NW, blowing a fresh breeze.

This one is dedicated to David Hedison.

(a) Seaview	120 gun 3 Decker SOL (elite crew) (43 pts)
(a) Flying Sub	40 gun Frigate (crack crew) (17 pts)
(b) Mermaid	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(s) Giant Squid	112 gun 3 Decker SOL (green crew) (27 pts)

Frigate Action:
Wind from the E, blowing a fresh breeze.

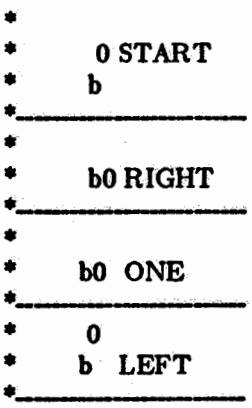
(a) Killdeer	40 gun Frigate (average crew) (15 pts)
(b) Sandpiper	40 gun Frigate (average crew) (15 pts)
(s) Curlew	38 gun Frigate (crack crew) (16 pts)

The Battle of Midway:
Wind from the E, blowing a moderate breeze.

(a) Enterprise	80 gun Ship of the Line (crack crew) (31 pts)
(a) Yorktown	80 gun Ship of the Line (average crew) (27 pts)
(a) Hornet	74 gun Ship of the Line (average crew) (24 pts)
(f) Akagi	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Kaga	96 gun 3 Decker SOL (green crew) (24 pts)
(f) Soryu	80 gun Ship of the Line (green crew) (23 pts)

EXAMPLE OF MOVE:

```
     / Max distance (including turns)
    / Max number of 45 degree turns (one at a time only)
   / /
Move(3, 2): rll    /* move right, ahead, left
```



SAMPLE GAME:

tutorial% sail
Choose a scenario:

NUMBER	SHIPS	IN PLAY	TITLE
0):	2	no	Ranger vs. Drake
1):	2	no	The Battle of Flamborough Head
2):	10	no	Arbuthnot and Des Touches
3):	10	no	Suffren and Hughes
4):	2	no	Nymphe vs. Cleopatre
5):	2	no	Mars vs. Hercule
6):	2	no	Ambuscade vs. Baionnaise
7):	2	no	Constellation vs. Insurgent
8):	2	no	Constellation vs. Vengeance
9):	10	no	The Battle of Lissa
10):	2	no	Constitution vs. Guerriere
11):	2	no	United States vs. Macedonian
12):	2	no	Constitution vs. Java
13):	2	no	Chesapeake vs. Shannon
14):	5	no	The Battle of Lake Erie
15):	2	no	Wasp vs. Reindeer
16):	3	no	Constitution vs. Cyane and Levant
17):	3	no	Pellew vs. Droits de L'Homme
18):	10	no	Algeciras
19):	7	no	Lake Champlain
20):	4	no	Last Voyage of the USS President
21):	2	no	Hornblower and the Natividad
22):	2	no	Curse of the Flying Dutchman
23):	4	no	The South Pacific
24):	5	no	Hornblower and the battle of Rosas bay
25):	5	no	Cape Horn
26):	3	no	New Orleans
27):	3	no	Botany Bay
28):	4	no	Voyage to the Bottom of the Sea
29):	3	no	Frigate Action
30):	6	no	The Battle of Midway

Scenario number? 21

Your ship is the Lydia, a 36 gun Frigate (elite crew).
Your name, Captain? **Dave #1**

Initial broadside left (grape, chain, round, double): d

Initial broadside right (grape, chain, round, double): r

```
Class 3 (36 guns) Frigate 'Lydia' (b0)        Points: 0  Fouls: 0  Grapples: 0
-------------------------------------------------------------------------------
|                                                                             |
|                                                                             |
|                                  ~0 -- a sinking ship                       |
|                 0                                                           |
|                 b                #1 -- an exploding ship                    |
|                 ^                                                           |
|                 bow of Lydia                  |-----------------|           |
|                                               |    wind speed  -5+|         |
|                                               |   and direction   |         |
|                                               | (blowing from right)|       |
|       ! -- a struck ship        S0            |-----------------|           |
|        1                        \                                           |
|                                 stern of Natividad                          |
|                                Natividad has full sails set.                |
|                                                                             |
-----------------------------Turn 0-------------------------------------------
Aye aye, Sir                    load: port and starboard - Load  D! R!    0 1(1)
~                                                 Hull  9      \|/
                                crew: 3 sections -------- Crew  4  4  2 -^-1(1)
                                guns: port and starboard - Guns  4  4    /|\
                              carronades: port and starboard --- Carr  2  2    | 3(5)
                                rigging 4 masts --------- Rigg 5 5 5 5   2(4)
```

NAME

snake, snscore – display chase game

SYNOPSIS

/usr/games/snake [**–w***n*] [**–l***n*]
/usr/games/snscore

DESCRIPTION

Snake is a display-based game which must be played on a CRT terminal from among those supported by vi(1). The object of the game is to make as much money as possible without getting eaten by the snake. The –l and –w options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as vi(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc These keys are like hjkl but form a directed pad around the d key.

HJKL These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.

SEFC Likewise for the upper case versions on the left.

ATPB These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, e.g. P is at the far right of the keyboard.

x This lets you quit the game at any time.

p Points in a direction you might want to go.

w Space warp to get out of tight squeezes, at a price.

! Shell escape

^Z Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run */usr/games/snscore* .

FILES

/usr/games/lib/snakerawscores database of personal bests
/usr/games/lib/snake.log log of games played

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

trek – trekkie game

SYNOPSIS

/usr/games/trek [[**–a**] file]

DESCRIPTION

Trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **–a** flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commadore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

COMMAND SUMMARY

abandon
cloak up/down
computer request; ...
destruct
help
lrscan
phasers automatic amount
phasers manual amt1 course1 spread1 ...
torpedo course [yes] angle/no
ram course distance
shell
srscan [yes/no]
status
undock
warp warp_factor
capture
damages
dock
impulse course distance
move course distance
rest time
shields up/down
terminate yes/no
visual course

NAME

worm – Play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In *worm,* you are a little worm, your body is the "o"'s on the screen and your head is the "@". You move with the hjkl keys (as in the game snake). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit, if your worm eats the digit is will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms – animate worms on a display terminal

SYNOPSIS

/usr/games/worms [–field] [–length #] [–number #] [–trail]

DESCRIPTION

Brian Horn (cithep!bdh) showed me a *TOPS-20* program on the DEC-2136 machine called *WORM*, and suggested that I write a similar program that would run under *Unix*. I did, and no apologies.

–field makes a "field" for the worm(s) to eat; **–trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

SEE ALSO

Snails, by Karl Heuer

BUGS

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump – the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

Wump **plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.**

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in ***People's Computer Company, 2,*** **2 (November 1973).**

NAME

miscellaneous – miscellaneous useful information pages

DESCRIPTION

This section contains miscellaneous documentation, mostly in the area of text processing macro packages for *troff*(1).

ascii	**map of ASCII character set**
eqnchar	**special character definitions for eqn**
hier	**file system hierarchy**
mailaddr	**mail addressing description**
man	**macros to typeset manual pages**
me	**macros for formatting papers**
ms	**macros for formatting manuscripts**

NAME

ascii – map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	'
050	(	051	)	052	*	053	+	054	,	055	-	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	'
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

FILES

/usr/pub/ascii

NAME

eqnchar – special character definitions for eqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] | **troff** [options]

neqn /usr/pub/eqnchar [files] | **nroff** [options]

DESCRIPTION

Eqnchar contains *troff* and *nroff* character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with *eqn* and *neqn*. It contains definitions for the following characters

ciplus	⊕	*||*	‖	*square*	
citimes	⊗	*langle*	⟨	*circle*	○
wig	∼	*rangle*	⟩	*blot*	
-wig	≈	*hbar*	ħ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<->*	↔	*empty*	∅
=wig	≅	*<=>*	⇔	*member*	∈
star	*	*|<*	≮	*nomem*	∉
bigstar	✱	*|>*	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	⊦	*incl*	⊂=
andsign	∧	*3dot*	∴	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊄
oppE	∃	*3quarter*	¾	*!supset*	⊅
angstrom	Å	*degree*	°		

FILES

/usr/pub/eqnchar

SEE ALSO

troff(1), eqn(1)

NAME

hier – file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative directory hierarchy.

/ root
/vmunix
the kernel binary (UNIX itself)
/lost+found
directory for connecting detached files for *fsck*(8)
/dev/ devices, see section 4
console main console, *tty*(4)
tty* terminals, *tty*(4)
xy* disks, *xy*(4S)
rxy* raw disks, *xy*(4S)
...
/bin/ utility programs, cf /usr/bin/ (described in sect. 1)
as assembler
cc C compiler executive, cf /lib/ccom, /lib/cpp, /lib/c2
csh C shell
...
/lib/ object libraries and other stuff, cf /usr/lib/
libc.a system calls, standard I/O, etc. (described in sect. 2, 3, 3S, 3C, 3N)
...
ccom C compiler proper
cpp C preprocessor
c2 C code improver
...
/etc/ essential data and maintenance utilities; described in section 8
dump dump program *dump*(8)
passwd password file, *passwd*(5)
group group file, *group*(5)
motd message of the day, *login*(1)
termcap
description of terminal capabilities, *termcap*(5)
ttytype table of what kind of terminal is on each port, *ttytype*(5)
mtab mounted file table, *mtab*(5)
dumpdates
dump history, *dump*(8)
fstab file system configuration table *fstab*(5)
ttys properties of terminals, *ttys*(5)
getty part of *login*, *getty*(8)
init the parent of all processes, *init*(8)
rc shell script to bring the system up
cron the clock daemon, *cron*(8)
mount *mount*(8)
wall *wall*(1)
...
/tmp/ temporary files, cf /usr/tmp/
e* used by *ed*(1)
ctm* used by *cc*(1)
...
/usr/ general-pupose directory, usually a mounted file system
adm/ administrative information

```
        wtmp    login history, utmp(5)
        messages
                hardware error messages
bin/    utility programs, to keep /bin/ small
etc/    administrative programs, to keep /etc/ small
tmp/    temporaries, to keep /tmp/ small
        stm*    used by sort(1)
dict/   word lists, etc.
        words   principal word list, used by look(1)
        spellhist
                history file for spell(1)
games/
        hangman
        lib/    library of stuff for the games
                quiz.k/ what quiz(6) knows
                        index   category index
                        africa  countries and capitals
                        ...
                ...
        ...
include/
        standard #include files
        a.out.h object file layout, a.out(5)
        stdio.h standard I/O, stdio(3S)
        math.h  (3M)
        ...
        sys/    system-defined layouts, cf /sys/h
lib/    object libraries and stuff, to keep /lib/ small
        atrun   scheduler for at(1)
        lint/   utility files for lint
                lint[12] subprocesses for lint(1)
                llib-lc  dummy declarations for /lib/libc.a, used by lint(1)
                llib-lm  dummy declarations for /lib/libc.m
                ...
        struct/ passes of struct(1)
        ...
        tmac/   macros for troff(1)
                tmac.an
                        macros for man(7)
                tmac.s  macros for ms(7)
                ...
        font/   fonts for troff(1)
                ftR     Times Roman
                ftB     Times Bold
                ...
        uucp/   programs and data for uucp(1C)
                L.sys   remote system names and numbers
                uucico  the real copy program
                ...
        units   conversion tables for units(1)
        eign    list of English words to be ignored by ptx(1)
```

```
/usr/   man/
        Pages for major manuals — User's Guide to Commands, UNIX Programmer's Manual,
        and System Manager's Guide.  man(1)
                man0/   general
                        intro     introduction to Sun System Manuals, in ms(7) format
                        xx        template for manual page
                man1/   chapter 1
                        as.1
                        spline.1g
                        ...
                ...
                cat1/   preformatted pages for section 1
                ...
        preserve/
                editor temporaries preserved here after crashes/hangups
        spool/  delayed execution files
                at/     used by at(1)
                lpd/    used by lpr(1)
                        lock      present when line printer is active
                        cf*       copy of file to be printed, if necessary
                        df*       daemon control file, lpd(8)
                        tf*       transient control file, while lpr is working
                uucp/   work files and staging area for uucp(1C)
                        LOGFILE
                                  summary log
                        LOG.*     log file for one transaction
                        ...
                mail/   mailboxes for mail(1)
                        name      mail file for user name
                        name.lock
                                  lock file while name is receiving mail
                secretmail/
                        like mail/
wd      initial working directory of a user, typically wd is the user's login name
        .profile  set environment for sh(1), environ(5)
        .cshrc    startup file for csh(1)
        .exrc     startup file for ex(1)
        .mailrc   startup file for mail(1)
        calendar
                user's datebook for calendar(1)
ucb/
binaries of programs developed at University of California at Berkeley.
...
edit    editor for beginners
ex      command editor for experienced users
...
mail    mail reading/sending subsystem
man     on line documentation
...
pi      Pascal translator
px      Pascal interpreter
...
vi      visual editor
```

SEE ALSO

ls(1), whatis(1), whereis(1), which (1), ncheck(8), find(1), grep(1)

BUGS

The position of files is subject to change without notice.

NAME

mailaddr – mail addressing description

DESCRIPTION

Mail addresses are based on the ARPANET protocols listed at the end of this manual page. These addresses are in the general format

user@domain

where a domain is a hierarchical dot separated list of subdomains. For example, the address

eric@monet.Berkeley.ARPA

is normally interpreted from right to left: the message should go to the ARPA name tables (which do not correspond exactly to the physical ARPANET), then to the Berkeley gateway, after which it should go to the local host monet. When the message reaches monet it is delivered to the user "eric".

Unlike some other forms of addressing, this does not imply any routing. Thus, although this address is specified as an ARPA address, it might travel by an alternate route if that was more convenient or efficient. For example, at Berkeley the associated message would probably go directly to monet over the Ethernet rather than going via the Berkeley ARPANET gateway.

Abbreviation. Under certain circumstances it may not be necessary to type the entire domain name. In general anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on "calder.Berkeley.ARPA" could send to "eric@monet" without adding the ".Berkeley.ARPA" since it is the same on both sending and receiving hosts.

Certain other abbreviations may be permitted as special cases. For example, at Berkeley ARPANET hosts can be referenced without adding the ".ARPA" as long as their names do not conflict with a local host name.

Compatibility. Certain old address formats are converted to the new format to provide compatibility with the previous mail system. In particular,

host:user

is converted to

user@host

to be consistent with the *rcp*(1C) command.

Also, the syntax:

host!user

is converted to:

user@host.UUCP

This is normally converted back to the "host!user" form before being sent on for compatibility with older UUCP hosts.

The current implementation is not able to route messages automatically through the UUCP network. This feature is planned for the 4.2 release. Until that time you must explicitly tell the mail system which hosts to send your message through to get to your final destination.

Case Distinctions. Domain names (i.e., anything after the "@" sign) may be given in any mixture of upper and lower case with the exception of UUCP hostnames. Most hosts accept any mixture of case in user names, with the notable exception of MULTICS sites.

Differences with ARPA Protocols. Although the UNIX addressing scheme is based on the ARPA mail addressing protocols, there are some significant differences.

At the time of this writing the only "top level" domain defined by ARPA is the ".ARPA" domain itself. This is further restricted to having only one level of host specifier. That is, the only addresses that ARPA accepts at this time must be in the format "user@host.ARPA" (where "host" is one word). In particular, addresses such as:

eric@monet.Berkeley.ARPA

are not currently legal under the ARPA protocols. For this reason, these addresses are converted to a different format on output to the ARPANET, typically:

eric%monet@Berkeley.ARPA

Route-addrs. Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this routing is done automatically, but sometimes it is desirable to route the message manually. An address that shows these relays are termed "route-addrs." These use the syntax:

<@hosta,@hostb:user@hostc>

This specifies that the message should be sent to hosta, from there to hostb, and finally to hostc. This path is forced even if there is a more efficient path to hostc.

Route-addrs occur frequently on return addresses, since these are generally augmented by the software at each host. It is generally possible to ignore all but the "user@host" part of the address to determine the actual sender.

Postmaster. Every site is required to have a user or user alias designated "postmaster" to which problems with the mail system may be addressed.

CSNET. Messages to CSNET sites can be sent to "user.host@UDel-Relay".

BERKELEY

The following comments apply only to the Berkeley environment.

Host Names. Many of the old familiar host names are being phased out. In particular, single character names as used in Berknet are incompatible with the larger world of which Berkeley is now a member. For this reason the following names are being obsoleted. You should notify any correspondents of your new address as soon as possible.

OLD	NEW			
p	ucbcad	j	ingvax	ucbingres
v csvax	ucbernie	r	arpavax	ucbarpa
n	ucbkim	y		ucbcory

The old addresses will be rejected as unknown hosts sometime in the near future.

What's My Address? If you are on a local machine, say monet, your address is

yourname@monet.Berkeley.ARPA

However, since most of the world does not have the new software in place yet, you will have to give correspondents slightly different addresses. From the ARPANET, your address would be:

yourname%monet@Berkeley.ARPA

From UUCP, your address would be:

ucbvax!monet.yourname

Computer Center. The Berkeley Computer Center is in a subdomain of Berkeley so that they may administer their own name space. Messages to the computer center should be addressed to one of:

user@host.CC.Berkeley.ARPA
user%host.CC@Berkeley.ARPA

depending on where the message is being sent from. The ".Berkeley.ARPA" may be omitted if the message is sent from inside Berkeley.

For the time being Computer Center hosts are known within the Berkeley domain, i.e., the ".CC" is optional. However, it is likely that this situation will change with time as both the Computer Science department and the Computer Center grow.

Bitnet. Hosts on bitnet may be accessed using:

user@host.BITNET

This works from 4.2 machines.

NAME

man – macros to typeset manual

SYNOPSIS

nroff –man file ...

troff –man file ...

DESCRIPTION

These macros are used to lay out pages of this manual. A skeleton page may be found in the file /usr/man/man0/xx.

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
/usr/man/man0/xx

SEE ALSO

troff(1), man(1)

BUGS

Relative indents don't nest.

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 c x v m*	yes	-	Begin page named *n* of chapter *c; x* is extra commentary, e.g. 'local', for

			page foot center; *v* alters page foot left, e.g. '4th Berkeley Distribution'; *m* alters page head center, e.g. 'Brand X Programmer's Manual'. Set prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Set prevailing indent to *i*. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

NAME

me – macros for formatting papers

SYNOPSIS

nroff –me [options] file ...
troff –me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, however these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

Output of the *eqn, neqn, refer,* and *tbl*(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1)
–me Reference Manual, Eric P. Allman
Writing Papers with Nroff Using –me

REQUESTS

In the following list, "initialization" refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The –me Reference Manual* for interesting details.

Request	Initial 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.
.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *'x'y'z'*	''''	no	Set even footer to x y z
.eh *'x'y'z'*	''''	no	Set even header to x y z
.fo *'x'y'z'*	''''	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *'x'y'z'*	''''	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x*; if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form .*x. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *'x'y'z'*	''''	no	Set odd footer to x y z
.oh *'x'y'z'*	''''	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

NAME

ms – text formatting macros

SYNOPSIS

nroff –ms [options] file ...
troff –ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through *col*(1). All external –ms macros are defined below.

Note that this –ms macro package is an extended version written at Berkeley and is a superset of the standard – ms macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed, for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many *nroff* and *troff* requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp n	insert n spacing lines
.ce n	center next n lines
.ls n	line spacing: n=1 single, n=2 double space
.na	no alignment of right margin

Font and point size changes with \f and \s are also allowed; for example, "\fIword\fR" will italicize *word*. Output of the *tbl*(1), *eqn*(1) and *refer*(1) preprocessors for equations, tables, and references is acceptable as input.

FILES

/usr/lib/tmac/tmac.s
/usr/lib/ms/ms.???

SEE ALSO

eqn(1), refer(1), tbl(1), troff(1)
Formatting Documents with the -ms Macro Package in
Editing and Text Processing on the Sun Workstation and the
Beginner's Guide to the Sun Workstation.

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	–	y	begin abstract; if *x*=no don't label abstract
.AE	–	y	end abstract
.AI	–	y	author's institution
.AM	–	n	better accent mark definitions
.AU	–	y	author's name
.B *x*	–	n	embolden *x*; if no *x*, switch to boldface
.B1	–	y	begin text to be enclosed in a box
.B2	–	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	–	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	–	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*

.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I,L,C,B; *y*=indent
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for .tl)
.EH *x*	–	n	even page header *x* (3 part as for .tl)
.EN	–	y	end displayed equation produced by *eqn*
.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for .tl)
.OH *x*	–	n	odd page header *x* (3 part as for .tl)
.P1	if TM	n	print header on 1st page
.PP	–	y,y	paragraph with first line indented
.PT	- -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on 1st page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set tabs to 8n 16n ... (nroff) 5n 10n ... (troff)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by *tbl*
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in *troff*
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; *y*=indent
.XE	–	y	end index entry (or series of .IX entries)
.XP	–	y,y	paragraph with first line exdented, others indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page

.2C	–	y,y	begin two column format
.]-	–	n	beginning of *refer* reference
.[0	–	n	end of unclassifiable type of reference
.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in –ms by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr LL 6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in –ms; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in *nroff*, `` in *troff*)
*U	unquote (" in *nroff*, '' in *troff*)
*-	dash (-- in *nroff*, — in *troff*)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*'	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.