UNIX User's Reference Manual
(URM)

4.3 Berkeley Software Distribution
Virtual VAX−11 Version

April, 1986

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720

This manual reflects system enhancements made at Berkeley and sponsored in part by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871 monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089. The views and conclusions contained in these documents are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency or of the US Government.

PREFACE

This update to the 4.2 distribution of August 1983 provides substantially improved performance, reliability, and security, the addition of Xerox Network System (NS) to the set of networking domains, and partial support for the VAX 8600 and MICROVAXII.

We were greatly assisted by the DEC UNIX Engineering group who provided two full time employees, Miriam Amos and Kevin Dunlap, to work at Berkeley. They were responsible for developing and debugging the distributed domain based name server and integrating it into the mail system. Mt Xinu provided the bug list distribution service as well as donating their MICROVAXII port to 4.3BSD. Drivers for the MICROVAXII were done by Rick Macklem at the University of Guelph. Sam Leffler provided valuable assistance and advice with many projects. Keith Sklower coordinated with William Nesheim and J. Q. Johnson at Cornell, and Chris Torek and James O'Toole at the University of Maryland to do the Xerox Network Systems implementation. Robert Elz at the University of Melbourne contributed greatly to the performance work in the kernel. Donn Seeley and Jay Lepreau at the University of Utah relentlessly dealt with a miriad of details; Donn completed the unfinished performance work on Fortran 77 and fixed numerous C compiler bugs. Ralph Campbell handled innumerable questions and problem reports and had time left to write rdist. George Goble was invaluable in shaking out the bugs on his production systems long before we were confident enough to inflict it on our users. Bill Shannon at Sun Microsystems has been helpful in providing us with bug fixes and improvements. Tom Ferrin, in his capacity as Board Member of Usenix Association, handled the logistics of large-scale reproduction of the 4.2BSD and 4.3BSD manuals. Mark Seiden helped with the typesetting and indexing of the 4.3BSD manuals. Special mention goes to Bob Henry for keeping ucbvax running in spite of new and improved software and an ever increasing mail, news, and uucp load.

Numerous others contributed their time and energy in creating the user contributed software for the release. As always, we are grateful to the UNIX user community for encouragement and support.

Once again, the financial support of the Defense Advanced Research Projects Agency is gratefully acknowledged.

M. K. McKusick
M. J. Karels
J. M. Bloom

Preface to the 4.2 Berkeley distribution

This update to the 4.1 distribution of June 1981 provides support for the VAX 11/730, full networking and interprocess communication support, an entirely new file system, and many other new features. It is certainly the most ambitious release of software ever prepared here and represents many man-years of work. Bill Shannon (both at DEC and at Sun Microsystems) and Robert Elz of the University of Melbourne contributed greatly to this distribution through new device drivers and painful debugging episodes. Rob Gurwitz of BBN wrote the initial version of the code upon which the current networking support is based. Eric Allman of Britton-Lee donated countless hours to the mail system. Bill Croft (both at SRI and Sun Microsystems) aided in the debugging and development of the networking facilities. Dennis Ritchie of Bell Laboratories also contributed greatly to this distribution, providing valuable advise and guidance. Helge Skrivervik worked on the device drivers which enabled the distribution to be delivered with a TU58 console cassette and RX01 console flopppy disk, and rewrote major portions of the standalone i/o system to support formatting of non-DEC peripherals.

Numerous others contributed their time and energy in organizing the user software for release, while many groups of people on campus suffered patiently through the low spots of development. As always, we are grateful to the UNIX user community for encouragement and support.

Once again, the financial support of the Defense Advanced Research Projects Agency is gratefully acknowledged.

S. J. Leffler
W. N. Joy
M. K. McKusick

Preface to the 4.1 Berkeley distribution

This update to the fourth distribution of November 1980 provides support for the VAX 11/750 and for the full interconnect architecture of the VAX 11/780. Robert Elz of the University of Melbourne contributed greatly to this distribution especially in the boot-time system configuration code; Bill Shannon of DEC supplied us with the implementation of DEC standard bad block handling. The research group at Bell Laboratories and DEC Merrimack provided us with access to 11/750's in order to debug its support.

Other individuals too numerous to mention provided us with bug reports, fixes and other enhancements which are reflected in the system. We are grateful to the UNIX user community for encouragement and support.

The financial support of the Defence Advanced Research Projects Agency in support of this work is gratefully acknowledged.

W. N. Joy
R. S. Fabry
K. Sklower

Preface to the Fourth Berkeley distribution

This manual reflects the Berkeley system mid-October, 1980. A large amount of tuning has been done in the system since the last release; we hope this provides as noticeable an improvement for you as it did for us. This release finds the system in transition; a number of facilities have been added in experimental versions (job control, resource limits) and the implementation of others is imminent (shared-segments, higher performance from the file system, etc.). Applications which use facilities that are in transition should be aware that some of the system calls and library routines will change in the near future. We have tried to be conscientious and make it very clear where this is likely.

A new group has been formed at Berkeley, to assume responsibility for the future development and support of a version of UNIX on the VAX. The group has received funding from the Defense Advanced Research Projects Agency (DARPA) to supply a standard version of the system to DARPA contractors. The same version of the system will be made available to other licensees of UNIX on the VAX for a duplication charge. We gratefully acknowledge the support of this contract.

We wish to acknowledge the contribution of a number of individuals to the the system.

We would especially like to thank Jim Kulp of IIASA, Laxenburg Austria and his colleagues, who first put job control facilities into UNIX; Eric Allman, Robert Henry, Peter Kessler and Kirk McKusick, who contributed major new pieces of software; Mark Horton, who contributed to the improvement of facilities and substantially improved the quality of our bit-mapped fonts, our hardware support staff: Bob Kridle, Anita Hirsch, Len Edmondson and Fred Archibald, who helped us to debug a number of new peripherals; Ken Arnold who did much of the leg-work in getting this version of the manual prepared, and did the final editing of sections 2-6, some special individuals within Bell Laboratories: Greg Chesson, Stuart Feldman, Dick Haight, Howard Katseff, Brian Kernighan, Tom London, John Reiser, Dennis Ritchie, Ken Thompson, and Peter Weinberger who helped out by answering questions; our excellent local DEC field service people, Kevin Althaus and Frank Chargois who kept our machine running virtually all the time, and fixed it quickly when things broke; and, Mike Accetta of Carnegie-Mellon University, Robert Elz of the University of Melbourne, George Goble of Purdue University, and David Kashtan of the Stanford Research Institute for their technical advice and support.

Special thanks to Bill Munson of DEC who helped by augmenting our computing facility and to Eric Allman for carefully proofreading the ''last'' draft of the manual and finding the bugs which we knew were there but couldn't see.

We dedicate this to the memory of David Sakrison, late chairman of our department, who gave his support to the establishment of our VAX computing facility, and to our department as a whole.

W. N. Joy
Ö. Babaoğlu
R. S. Fabry
K. Sklower

Preface to the Third Berkeley distribution

This manual reflects the state of the Berkeley system, December 1979. We would like to thank all the people at Berkeley who have contributed to the system, and particularly thank Prof. Richard Fateman for creating and administrating a hospitable environment, Mark Horton who helped prepare this manual, and Eric Allman, Bob Kridle, Juan Porcar and Richard Tuck for their contributions to the kernel.

The cooperation of Bell Laboratories in providing us with an early version of UNIX/32V is greatly appreciated. We would especially like to thank Dr. Charles Roberts of Bell Laboratories for helping us obtain this release, and acknowledge T. B. London, J. F. Reiser, K. Thompson, D. M. Ritchie, G. Chesson and H. P. Katseff for their advice and support.

W. N. Joy
Ö. Babaoğlu

Preface to the UNIX/32V distribution

The UNIX† operating system for the VAX*-11 provides substantially the same facilities as the UNIX system for the PDP*-11.

We acknowledge the work of many who came before us, and particularly thank G. K. Swanson, W. M. Cardoza, D. K. Sharma, and J. F. Jarvis for assistance with the implementation for the VAX-11/780.

T. B. London
J. F. Reiser

Preface to the Seventh Edition

Although this Seventh Edition no longer bears their byline, Ken Thompson and Dennis Ritchie remain the fathers and preceptors of the UNIX time-sharing system. Many of the improvements here described bear their mark. Among many, many other people who have contributed to the further flowering of UNIX, we wish especially to acknowledge the contributions of A. V. Aho, S. R. Bourne, L. L. Cherry, G. L. Chesson, S. I. Feldman, C. B. Haley, R. C. Haight, S. C. Johnson, M. E. Lesk, T. L. Lyon, L. E. McMahon, R. Morris, R. Muha, D. A. Nowitz, L. Wehr, and P. J. Weinberger. We appreciate also the effective advice and criticism of T. A. Dolotta, A. G. Fraser, J. F. Maranzano, and J. R. Mashey; and we remember the important work of the late Joseph F. Ossanna.

B. W. Kernighan
M. D. McIlroy

† UNIX is a registered trademark of AT&T Bell Laboratories in the USA and other countries.
*VAX and PDP are Trademarks of Digital Equipment Corporation.

INTRODUCTION TO USER'S REFERENCE MANUAL

The documentation has been reorganized for 4.3BSD in a format similar to the one used for the Usenix 4.2BSD manuals. It is divided into three sets; each set consists of one or more volumes. The abbreviations for the volume names are listed in square brackets; the abbreviations for the manual sections are listed in parenthesis.

I. User's Documents
- User's Reference Manual [URM]
  - Commands (1)
  - Games (6)
  - Macro packages and language conventions (7)
- User's Supplementary Documents [USD]
  - Getting Started
  - Basic Utilities
  - Communicating with the World
  - Text Editing
  - Document Preparation
  - Amusements

II. Programmer's Documents
- Programmer's Reference Manual [PRM]
  - System calls (2)
  - Subroutines (3)
  - Special files (4)
  - File formats and conventions (5)
- Programmer's Supplementary Documents, Volume 1 [PS1]
  - Languages in common use
  - General Reference
  - Programming Tools
  - Programming Libraries
- Programmer's Supplementary Documents, Volume 2 [PS2]
  - Documents of Historic Interest
  - Other Languages
  - Database Management

III. System Manager's Manual [SMM]
- Maintenance commands (8)
- System Installation and Administration
- Supporting Documentation

References to individual documents are given as ''volume:document'', thus USD:1 refers to the first document in the ''User's Supplementary Documents''. References to manual pages are given as ''*name*(section)'' thus *sh*(1) refers to the shell manual entry in section 1.

The manual pages give descriptions of the publicly available features of the UNIX/32V† system, as extended to provide a virtual memory environment and other enhancements at the University of California. They do not attempt to provide perspective or tutorial information about the UNIX operating system, its facilities, or its

† UNIX is a registered trademark of AT&T Bell Laboratories in the USA and other countries.

implementation. Various documents on those topics are contained in the ‘‘UNIX User’s Supplementary Documents’’ (USD), the ‘‘UNIX Programmer’s Supplementary Documents’’ (PS1 and PS2), and ‘‘UNIX System Manager’s Manual’’ (SMM). In particular, for an overview see ‘‘The UNIX Time-Sharing System’’ (PS2:1) by Ritchie and Thompson; for a tutorial see ‘‘UNIX for Beginners’’ (USD:1) by Kernighan, and for an guide to the new features of this virtual version, see ‘‘Berkeley Software Architecture Manual (4.3 Edition)’’ (PS1:6).

Within the area it surveys, this volume attempts to be timely, complete and concise. Where the latter two objectives conflict, the obvious is often left unsaid in favor of brevity. It is intended that each program be described as it is, not as it should be. Inevitably, this means that various sections will soon be out of date.

Commands are programs intended to be invoked directly by the user, in contrast to subroutines, that are intended to be called by the user’s programs. User commands are described in URM section 1. Commands generally reside in directory */bin* (for *bin* ary programs). Some programs also reside in */usr/bin, /usr/ucb,* or */usr/new,* to save space in */bin.* These directories are searched automatically by the command interpreters.

Games have been relegated to URM section 6 and */usr/games,* to keep them from contaminating the more staid information of URM section 1.

Miscellaneous collection of information necessary for writing in various specialized languages such as character codes, macro packages for typesetting, etc is contained in URM section 7.

System calls are entries into the UNIX supervisor. The system call interface is identical to a C language procedure call; the equivalent C procedures are described in PRM section 2.

An assortment of subroutines is available; they are described in PRM section 3. The primary libraries in which they are kept are described in *intro*(3). The functions are described in terms of C; those that will work with Fortran are described in *intro*(3f).

PRM section 4 discusses the characteristics of each system ‘‘file’’ that refers to an I/O device. The names in this section refer to the DEC device names for the hardware, instead of the names of the special files themselves.

The file formats and conventions (PRM section 5) documents the structure of particular kinds of files; for example, the form of the output of the loader and assembler is given. Excluded are files used by only one command, for example the assembler’s intermediate files.

Commands and procedures intended for use primarily by the system administrator are described in SMM section 8. The commands and files described here are almost all kept in the directory */etc.*

Each section consists of independent entries of a page or so each. The name of the entry is in the upper corners of its pages, together with the section number, and sometimes a letter characteristic of a subcategory, e.g. graphics is 1G, and the math library is 3M. Entries within each section are alphabetized. except for PRM section 3f which appears after the rest of PRM section 3. The page numbers of each entry start at 1; it is infeasible to number consecutively the pages of a document like this that is republished in many variant forms.

All entries are based on a common format; not all subsections always appear.

> The *name* subsection lists the exact names of the commands and subroutines covered under the entry and gives a short description of their purpose.
>
> The *synopsis* summarizes the use of the program being described. A few conventions are used, particularly in the Commands subsection:
>
> > **Boldface** words are considered literals, and are typed just as they appear.
> >
> > Square brackets [ ] around an argument show that the argument is optional. When an argument is given as ‘‘name’’, it always refers to a file name.
> >
> > Ellipses ‘‘. . .’’ are used to show that the previous argument-prototype may be repeated.
> >
> > A final convention is used by the commands themselves. An argument beginning with a minus sign ‘‘−’’ usually means that it is an option-specifying argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with ‘‘−’’.
>
> The *description* subsection discusses in detail the subject at hand.
>
> The *files* subsection gives the names of files that are built into the program.

A *see also* subsection gives pointers to related information.

A *diagnostics* subsection discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The *bugs* subsection gives known bugs and sometimes deficiencies. Occasionally the suggested fix is also described.

At the beginning of URM is a table of contents, organized by section and alphabetically within each section. There is also a permuted index derived from the table of contents. Within each index entry, the title of the writeup to which it refers is followed by the appropriate section number in parentheses. This fact is important because there is considerable name duplication among the sections, arising principally from commands that exist only to exercise a particular system call.

HOW TO GET STARTED

This section sketches the basic information you need to get started on UNIX; how to log in and log out, how to communicate through your terminal, and how to run a program. See ‘‘UNIX for Beginners’’ in (USD:1) for a more complete introduction to the system.

Logging in. Almost any ASCII terminal capable of full duplex operation and generating the entire character set can be used. You must have a valid user name, which may be obtained from the system administration. If you will be accessing UNIX remotely, you will also need to obtain the telephone number for the system that you will be using.

After a data connection is established, the login procedure depends on what type of terminal you are using and local system conventions. If your terminal is directly connected to the computer, it generally runs at 9600 or 19200 baud. If you are using a modem running over a phone line, the terminal must be set at the speed appropriate for the modem you are using, typically 300, 1200, or 2400 baud. The half/full duplex switch should always be set at full-duplex. (This switch will often have to be changed since many other systems require half-duplex).

When a connection is established, the system types ‘‘login:’’; you type your user name, followed by the ‘‘return’’ key. If you have a password, the system asks for it and suppresses echo to the terminal so the password will not appear. After you have logged in, the ‘‘return’’, ‘‘new line’’, or ‘‘linefeed’’ keys will give exactly the same results. A message-of-the-day usually greets you before your first prompt.

If the system types out a few garbage characters after you have established a data connection (the ‘‘login:’’ message at the wrong speed), depress the ‘‘break’’ (or ‘‘interrupt’’) key. This is a speed-independent signal to UNIX that a different speed terminal is in use. The system then will type ‘‘login:,’’ this time at another speed. Continue depressing the break key until ‘‘login:’’ appears clearly, then respond with your user name.

For all these terminals, it is important that you type your name in lower-case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and will translate all subsequent lower-case letters to upper case.

The evidence that you have successfully logged in is that a shell program will type a prompt (‘‘$’’ or ‘‘%’’) to you. (The shells are described below under ‘‘How to run a program.’’)

For more information, consult *tset*(1), and *stty*(1), which tell how to adjust terminal behavior; *getty*(8) discusses the login sequence in more detail, and *tty*(4) discusses terminal I/O.

Logging out. There are three ways to log out:

By typing ‘‘logout’’ or an end-of-file indication (EOT character, control-D) to the shell. The shell will terminate and the ‘‘login:’’ message will appear again.

You can log in directly as another user by giving a *login*(1) command.

If worse comes to worse, you can simply hang up the phone; but beware − some machines may lack the necessary hardware to detect that the phone has been hung up. Ask your system administrator if this is a problem on your machine.

How to communicate through your terminal. When you type characters, a gnome deep in the system gathers your characters and saves them in a secret place. The characters will not be given to a program until you type a return (or newline), as described above in *Logging in.*

UNIX terminal I/O is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the printed output will have the input characters interspersed. However, whatever you type will be saved up and interpreted in correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away all the saved characters (or beeps, if your prompt was a ‘‘%’’).

The delete (DEL) character in typed input kills all the preceding characters in the line, so typing mistakes can be repaired on a single line. Also, the backspace character (control-H) erases the last character typed. *Tset*(1) or *stty*(1) can be used to change these defaults. Successive uses of backspace erases characters back to, but not beyond, the beginning of the line. DEL and backspace can be transmitted to a program by preceding them with ‘‘\’’. (So, to erase ‘‘\’’, you need two backspaces).

An *interrupt signal* is sent to a program by typing control-C or the ‘‘break’’ key which is not passed to programs. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you do not want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor, for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited. The interrupt character can also be changed with *tset*(1) or *stty*(1).

It is also possible to suspend output temporarily using ^S (control-S) and later resume output with ^Q (control-Q). Output can be thrown away without interrupting the program by typing ^O (control-O); see *tty*(4).

The *quit* signal is generated by typing the ASCII FS character. (FS appears many places on different terminals, most commonly as control-\ or control-|.) It not only causes a running program to terminate but also generates a file with the core image of the terminated process. Quit is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent about whether you have a terminal with the newline function or whether it must be simulated with carriage-return and line-feed. In the latter case, all input carriage returns are turned to newline characters (the standard line delimiter) and both a carriage return and a line feed are echoed to the terminal. If you get into the wrong mode, the *reset*(1) command will rescue you. If the terminal does not appear to be echoing anything that you type, it may be stuck in ‘‘no-echo’’ or ‘‘raw’’ mode. Try typing ‘‘(control-J)reset(control-J)’’ to recover.

Tab characters are used freely in UNIX source programs. If your terminal does not have the tab function, you can arrange to have them turned into spaces during output, and echoed as spaces during input. The system assumes that tabs are set every eight columns. Again, the *tset*(1) or *stty*(1) command can be used to change these defaults. *Tset*(1) can be used to set the tab stops automatically when necessary.

How to run a program; the shells. When you have successfully logged in, a program called a shell is listening to your terminal. The shell reads typed-in lines, splits them up into a command name and arguments, and executes the command. A command is simply an executable program. The shell looks in several system directories to find the command. You can also place commands in your own directory and have the shell find them there. There is nothing special about system-provided commands except that they are kept in a directory where the shell can find them.

The command name is always the first word on an input line; it and its arguments are separated from one another by spaces.

When a program terminates, the shell will ordinarily regain control and type a prompt at you to show that it is ready for another command.

The shells have many other capabilities, that are described in detail in sections *sh*(1) and *csh*(1). If the shell prompts you with ‘‘$’’, then it is an instance of *sh*(1) the standard shell provided by Bell Labs. If it prompts with ‘‘%’’ then it is an instance of *csh*(1), a shell written at Berkeley. The shells are different for all but the most simple terminal usage. Most users at Berkeley choose *csh*(1) because of the *history* mechanism and the *alias* feature, that greatly enhance its power when used interactively. *Csh* also supports the job-control facilities; see *csh*(1) or the Csh introduction in USD:4 for details.

You can change from one shell to the other by using the *chsh*(1) command, which takes effect at your next login.

The current directory. UNIX has a file system arranged as a hierarchy of directories. When the system administrator gave you a user name, they also created a directory for you (ordinarily with the same name as your user name). When you log in, any file name you type is by default in this directory. Since you are the owner of this directory, you have full permission to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their owners. As a matter of observed fact, few UNIX users protect their files from perusal by other users.

To change the current directory (but not the set of permissions you were endowed with at login) use *cd*(1).

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with ''/'', the name of the root directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a ''/'') until finally the file name is reached. For example, */usr/tmp/filex* refers to the file *filex* in the directory *tmp; tmp* is itself a subdirectory of *usr; usr* springs directly from the root directory.

If your current directory has subdirectories, the path names of files therein begin with the name of the subdirectory with no prefixed ''/''.

A path name may be used anywhere a file name is required.

Important commands that modify the contents of files are *cp*(1), *mv*(1), and *rm*(1), which respectively copy, move (i.e. rename) and remove files. To find out the status of files or directories, use *ls*(1). See *mkdir*(1) for making directories and *rmdir*(1) for destroying them.

For a fuller discussion of the file system, see ''A Fast File System for UNIX'' (SMM:14) by McKusick, Joy, Leffler, and Fabry. It may also be useful to glance through PRM section 2, that discusses system calls, even if you do not intend to deal with the system at that level.

Writing a program. To enter the text of a source program into a UNIX file, use the editor *ex*(1) or its display editing alias *vi*(1). (The old standard editor *ed*(1) is also available.) The principal languages in UNIX are provided by the C compiler *cc*(1), the Fortran compiler *f77*(1), and its derivatives *efl*(1) and *ratfor*(1), the Pascal compiler *pc*(1), and interpreter *pi*(1), and the Lisp system *lisp*(1). User contributed software in the latest release of the system supports APL, B, the Functional Programming language, and Icon. Refer to *apl*(1), *b*(1), *fp*(1), and *icon*(1), respectively for more information about each. After the program text has been entered through the editor and written to a file, you can give the file to the appropriate language processor as an argument. The output of the language processor will be left on a file in the current directory named ''a.out''. If the output is precious, use *mv*(1) to move it to a less exposed name after successful compilation.

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the shell in response to the shell (''$'' or ''%'') prompt.

Your programs can receive arguments from the command line just as system programs do, see ''UNIX Programming - Second Edition'' (PS2:3), or for a more terse description *execve*(2).

Text processing. Almost all text is entered through the editor *ex*(1) (often entered via *vi*(1)). The commands most often used to write text on a terminal are: *cat*(1), *more*(1), and *nroff*(1).

The *cat*(1) command simply dumps ASCII text on the terminal, with no processing at all. *More*(1) is useful for preventing the output of a command from scrolling off the top of your screen. It is also well suited to perusing files. *Nroff*(1) is an elaborate text formatting program. Used naked, it requires careful forethought, but for ordinary documents it has been tamed; see *me*(7) and *ms*(7).

Troff(1) prepares documents for a Graphics Systems phototypesetter or a Versatec Plotter; it is similar to *nroff*(1), and often works from exactly the same source text. It was used to produce this manual.

Script(1) lets you keep a record of your session in a file, which can then be printed, mailed, etc. It provides the advantages of a hard-copy terminal even when using a display terminal.

Status inquiries. Various commands exist to provide you with useful information. *w*(1) prints a list of users currently logged in, and what they are doing. *date*(1) prints the current time and date. *ls*(1) will list the files in your directory or give summary information about particular files.

Surprises. Certain commands provide inter-user communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you.

To communicate with another user currently logged in, *write*(1) or *talk*(1) is used; *mail*(1) will leave a message whose presence will be announced to another user when they next log in. The write-ups in the manual also suggest how to respond to the these commands if you are a target.

If you use *csh*(1) the key ^Z (control-Z) will cause jobs to ''stop''. If this happens before you learn about it, you can simply continue by saying ''fg'' (for foreground) to bring the job back.

CONVERTING FROM 4.2BSD SYSTEMS

A detailed list of changes from the 4.2BSD to the 4.3BSD distributions is contained in ''Bug Fixes and Changes in 4.3BSD'' (SMM:12), and ''Changes to the Kernel in 4.3BSD'' (SMM:13). Detailed conversion procedures are described in ''Installing and Operating 4.3BSD on the VAX'' (SMM:1); it also discusses changes from pre-4.2BSD systems.

TABLE OF CONTENTS

1. Commands and Application Programs

2. System Calls

3. C Library Subroutines

3F. Fortran Library

4. Special Files

5. File Formats

6. Games

7. Miscellaneous

8. System Maintenance

PERMUTED INDEX

NAME

intro − introduction to commands

DESCRIPTION

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

(1) Commands of general utility.

(1C) Commands for communication with other systems.

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

N.B.: Commands related to system maintenance used to appear in section 1 manual pages and were distinguished by (1M) at the top of the page. These manual pages now appear in section 8.

SEE ALSO

Section (6) for computer games.

How to get started, in the Introduction.

DIAGNOSTICS

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

NAME

adb − debugger

SYNOPSIS

adb [**−w**] [**−k**] [**-I**dir] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. 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.**

Requests to *adb* are read from the standard input and responses are to the standard output. If the **−w** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*.

The **−k** option makes *adb* do UNIX kernel memory mapping; it should be used when *core* is a UNIX crash dump or */dev/mem*.

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

Adb ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

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

If *address* is present then *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. If the operating system is being debugged either post-mortem or using the special file */dev/mem* to interactive examine and/or modify memory the maps are set to map the kernel virtual addresses which start at 0x80000000 (on the VAX). ADDRESSES.

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.

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. \ 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 utter 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*. (This form is currently broken on the VAX; local variables can be examined only with *dbx*(1).)

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

Monadic operators

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

@*exp* The contents of the location addressed by *exp* in *objfil*.

−*exp* Integer negation.

~*exp* Bitwise complement.

#*exp* Logical negation.

Dyadic operators are left associative and are less binding than monadic 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*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '*'; see ADDRESSES for further details.)

?*f* Locations starting at *address* in *objfil* are printed according to the format *f*. *dot* is incremented by the sum of the increments for each format letter (q.v.).

/*f* Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for '?'.

=*f* The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format '?' is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

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 where control characters are printed as ^X and the delete character is printed 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. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
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**.
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.

newline Repeat the previous command with a *count* of 1.

[**?/**]**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 mapping is changed. 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*.)

>*name* *Dot* is assigned to the variable or register named.

! A shell (/bin/sh) is called to read the rest of the line following '!'.

$*modifier*

Miscellaneous commands. The available *modifiers* are:

- **<***f* Read commands from the file *f*. If this command is executed in a file, further commands in the file are not seen. If *f* 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 *f* is executed.
- **<<***f* Similar to **<** except it can be used in a file of commands without causing the file to be closed. 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.
- **>***f* Append output to the file *f*, which is created if it does not exist. If *f* is omitted, output is returned to the terminal.
- **?** Print process id, the signal which caused stoppage or termination, as well as the registers as **$r**. This is the default if *modifier* is omitted.
- **r** Print the general registers and the instruction addressed by **pc**. *Dot* is set to **pc**.
- **b** Print all breakpoints and their associated counts and commands.
- **c** C stack backtrace. If *address* is given then it is taken as the address of the current frame instead of the contents of the frame−pointer register. If **C** is used then the names and (32 bit) values of all automatic and static variables are printed for each active function. (broken on the VAX). If *count* is given then only the first *count* frames are printed.
- **d** Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus ''10$d'' never changes the default radix. To make decimal the default radix, use ''0t10$d''.
- **e** The names and values of external variables are printed.
- **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.
- **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 user page table entries (on the VAX).

:*modifier*

Manage a subprocess. Available modifiers are:

- **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.
- **d** Delete breakpoint at *address*.
- **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.
- **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.

k The current subprocess, if any, is terminated.

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.

ADDRESSES

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.

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

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

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. 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)

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

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

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), lookbib(1)

AUTHORS

Al Stangenberger, Bill Tuthill

NAME

apply − apply a command to a set of arguments

SYNOPSIS

apply [**−a***c*] [**−***n*] command args ...

DESCRIPTION

Apply runs the named *command* on each argument *arg* in turn. Normally arguments are chosen singly; the optional number *n* specifies the number of arguments to be passed to *command.* If *n* is zero, *command* is run without arguments once for each *arg*. Character sequences of the form *%d* in *command,* where *d* is a digit from 1 to 9, are replaced by the *d*'th following unused *arg*. If any such sequences occur, *n* is ignored, and the number of arguments passed to *command* is the maximum value of *d* in *command.* The character '%' may be changed by the **−a** option.

Examples:

    apply echo *

is similar to ls(1);

    apply −2 cmp a1 b1 a2 b2 ...

compares the 'a' files to the 'b' files;

    apply −0 who 1 2 3 4 5

runs who(1) 5 times; and

    apply ´ln %1 /usr/joe´ *

links all files in the current directory to the directory /usr/joe.

SEE ALSO

sh(1)

AUTHOR

Rob Pike

BUGS

Shell metacharacters in *command* may have bizarre effects; it is best to enclose complicated commands in single quotes ´ ´.

There is no way to pass a literal '%2' if '%' is the argument expansion character.

NAME

apropos − locate commands by keyword lookup

SYNOPSIS

apropos keyword ...

DESCRIPTION

Apropos shows which manual sections contain instances of any of the given keywords in their title. Each word is considered separately and case of letters is ignored. Words which are part of other words are considered; thus, when looking for compile, *apropos* will find all instances of 'compiler' also. Try

apropos password

and

apropos editor

If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'apropos format' and then 'man 3s printf' to get the manual on the subroutine *printf.*

Apropos is actually just the **−k** option to the *man*(1) command.

FILES

/usr/man/whatis data base

SEE ALSO

man(1), whatis(1), catman(8)

AUTHOR

William Joy

NAME

ar − archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the loader. It can be used, though, for any similar purpose. **N.B:** This version of *ar* uses a ASCII-format archive which is portable among the various machines running UNIX. Programs for dealing with older formats are available: see *arcv*(8).

Key is one character from the set **drqtpmx,** optionally concatenated with one or more of **vuaibclo.** *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r,** then only those files with 'last-modified' dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r,** specifies where the files are to be moved.

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. Normally the 'last-modified' date of each extracted file is the date when it is extracted. However, if **o** is used, the 'last-modified' date is reset to the date recorded in the archive.

v Verbose. Under the verbose option, *ar* gives a file-by-file description of the making 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 Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally *ar* places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

FILES

/tmp/v* temporaries

SEE ALSO

lorder(1), ld(1), ranlib(1), ar(5), arcv(8)

BUGS

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

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

NAME

as − VAX-11 assembler

SYNOPSIS

as [**−a1-16**] [[**−d124**] [**−L**] [**−W**] [**−V**] [**−J**] [**−R**] [**−t** directory] [**−o** objfile] [name ...]

DESCRIPTION

As assembles the named files, or the standard input if no file name is specified. The available flags are:

−a Specifies the alignment of procedures and data blocks. It is given as a power of two; thus an alignment of 3 causes alignment on an eight byte boundary. The default is **−a2.**

−d Specifies the number of bytes to be assembled for offsets which involve forward or external references, and which have sizes unspecified in the assembly language. The default is **−d4.**

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

−V Use virtual memory for some intermediate storage, rather than a temporary file.

−W Do not complain about errors.

−J Use long branches to resolve jumps when byte-displacement branches are insufficient. This must be used when a compiler-generated assembly contains branches of more than 32k bytes.

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

−t Specifies a directory to receive the temporary file, other than the default /tmp.

All undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file *objfile;* if that is omitted, *a.out* is used.

FILES

/tmp/as*	default temporary files
a.out	default resultant object file

SEE ALSO

ld(1), nm(1), adb(1), dbx(1), a.out(5)
Auxiliary documentation *Assembler Reference Manual.*

AUTHORS

John F. Reiser
Robert R. Henry

BUGS

−J should be eliminated; the assembler should automatically choose among byte, word and long branches.

NAME

at − execute commands at a later time

SYNOPSIS

at [-c] [-s] [-m] time [day] [file]

DESCRIPTION

At spools away a copy of the named *file* to be used as input to *sh*(1) or *csh*(1). If the **−c** flag (for (*csh*(1))) or the **−s** flag (for (*sh*(1))) is specified, then that shell will be used to execute the job; if no shell is specified, the current environment shell is used. If no file name is specified, *at* prompts for commands from standard input until a ^D is typed.

If the **−m** flag is specified, mail will be sent to the user after the job has been run. If errors occur during execution of the job, then a copy of the error diagnostics will be sent to the user. If no errors occur, then a short message is sent informing the user that no errors occurred.

The format of the spool file is as follows: A four line header that includes the owner of the job, the name of the job, the shell used to run the job, and whether mail will be set after the job is executed. The header is followed by a *cd* command to the current directory and a *umask* command to set the modes on any files created by the job. Then *at* copies all relevant environment variables to the spool file. When the script is run, it uses the user and group ID of the creator of the spool file.

The *time* is 1 to 4 digits, with an optional following 'A', 'P', 'N' or 'M' for AM, PM, noon or midnight. 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.

The optional *day* is either (1) a month name followed by a day number, or (2) a day of the week; if the word 'week' follows, invocation is moved seven days further off. Names of months and days may be recognizably truncated. Examples of legitimate commands are

```
at 8am jan 24
at -c -m 1530 fr week
at -s -m 1200n week
```

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

Error output is lost unless redirected or the *−m* flag is requested, in which case a copy of the errors is sent to the user via *mail*(1).

FILES

/usr/spool/at	spooling area
/usr/spool/at/yy.ddd.hhhh.*	job file
/usr/spool/at/past	directory where jobs are executed from
/usr/spool/at/lasttimedone	last time atrun was run
/usr/lib/atrun	executor (run by cron(8))

SEE ALSO

atq(1), atrm(1), calendar(1), sleep(1), cron(8)

DIAGNOSTICS

Complains about various syntax errors and times out of range.

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.

If the system crashes, mail is not sent to the user informing them that the job was not completed.

Sometimes old spool files are not removed from the directory /usr/spool/at/past. This is usually due to a system crash, and requires that they be removed by hand.

NAME

atq − print the queue of jobs waiting to be run

SYNOPSIS

atq [-c] [-n] [name ...]

DESCRIPTION

Atq prints the queue of jobs that are waiting to be run at a later date. These jobs were created with the *at*(1) command. With no flags, the queue is sorted in the order that the jobs will be executed.

If the **−c** flag is used, the queue is sorted by the time that the *at* command was given.

The **−n** flag prints only the total number of files that are currently in the queue.

If a name(s) is provided, only those files belonging to that user(s) are displayed.

FILES

/usr/spool/at spool area

SEE ALSO

at(1), atrm(1), cron(8)

NAME

atrm − remove jobs spooled by at

SYNOPSIS

atrm [-f] [-i] [-] [[job #] [name]...]

DESCRIPTION

Atrm removes jobs that were created with the *at*(1) command. With the **−** flag, all jobs belonging to the person invoking *atrm* are removed. If a job number(s) is specified, *atrm* attempts to remove only that job number(s).

If the **−f** flag is used, all information regarding the removal of the specified jobs is suppressed. If the **−i** flag is used, *atrm* asks if a job should be removed; a response of 'y' causes the job to be removed.

If a user(s) name is specified, all jobs belonging to that user(s) are removed. This form of invoking *atrm* is useful only to the super-user.

FILES

/usr/spool/at spool area

SEE ALSO

at(1), atq(1), cron(8)

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [**−F***c*] [prog] [file] ...

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog,* or in a file specified as **−f** *file*.

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

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

A pattern-action statement has the form

```
pattern { action }
```

A missing { action } means print the line; a missing pattern always matches.

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

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

Statements are terminated by semicolons, 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; this allows for a form of associative memory. String constants are quoted "...".

The *print* statement prints its arguments on the standard output (or on a file if *>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 (see *printf*(3S)).

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

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

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

A relational expression is one of the following:

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

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

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

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

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

or by using the **−F***c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default 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)
A. V. Aho, B. W. Kernighan, P. J. Weinberger, *Awk − a pattern scanning and processing language*

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 prints the result on the standard output. It is normally used inside substitution marks \` \` in shell procedures.

This shell procedure invoked with the argument */usr/src/bin/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. It takes input from any files given, then reads the standard input. The **−l** argument stands for the name of an arbitrary precision math library. The syntax for *bc* programs is as follows; L means letter a-z, E means expression, S means statement.

Comments
    are enclosed in /* and */.

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

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

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

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

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

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

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or 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.

For example

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

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

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

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

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

bib, listrefs − bibliographic formatter; list bibliographic reference items

SYNOPSIS

bib [options] ...
listrefs [options] ...

DESCRIPTION

Bib is a preprocessor for *nroff* or *troff*(1) that formats citations and bibliographies. The input files (standard input default) are copied to the standard output, except for text between [. and .] pairs, which are assumed to be keywords for searching a bibliographic database. If a matching reference is found a citation is generated replacing the text. References are collected, optionally sorted, and written out at a location specified by the user. Citation and reference formats are controlled by the −t option.

Reference databases are created using the *invert* utility.

The following options are available. Note that standard format styles (see the −t option) set options automatically. Thus if a standard format style is used the user need not indicate any further options for most documents.

−aa reduce authors first names to abbreviations.

−ar*num* reverse the first *num* author's names. If a number is not given all authors names are reversed.

−ax print authors last names in Caps-Small Caps style. For example Budd becomes BUDD. This style is used by certain ACM publications.

−c*str* build citations according to the template *str*. See the reference format designers guide for more information on templates.

−ea reduce editors first names to abbreviations.

−ex print editors last names in Caps-Small Caps style (see −ax option).

−er*num* reverse the first *num* editors names. If a number is not given all editors names are reversed.

−f instead of collecting references, dump each reference immediately following the line on which the citation is placed (used for footnoted references).

−i *file*

−i*file* process the indicated file, such as a file of definitions. (see technical report for a description of file format).

−h replace citations to three or more adjacent reference items with a hyphenated string (eg 2,3,4,5 becomes 2-5). This option implies the −o option.

−n*str* turn off indicated options. *str* must be composed of the letters afhosx.

−o contiguous citations are ordered according the the reference list before being printed (default).

−p *file*

−p*file* instead of searching the file INDEX, search the indicated reference files before searching the system file. *files* is a comma separated list of inverted indices, created using the *invert* utility.

−s*str* sort references according to the template *str*.

−t *type*

−t*type* use the standard macros and switch settings for the indicated style to generate citations and references. There are a number of standard styles provided. In addition users can generate their own style macros. See the format designers guide for details.

Listrefs formats an entire reference database file. Options to *listrefs* are the same as for *bib*.

Bib was designed initially for use with the −ms macros, and uses a couple of the −ms macros (.ip and .lp) in its macro definitions. To use it with the −me macros, prefix the file being sent to nroff/troff with the following macro definitions:

```
.de IP
.ip \$1 \$2
..
.de LP
.lp
..
```

A file "bibmac.me" containing these macro definitions may be found in /usr/new/lib/bmac.

FILES

INDEX	inverted index for reference database
/usr/dict/papers/INDEX	default system index
/usr/new/lib/bmac/bmac.*	formatting macro packages
/usr/tmp/bibr*	scratch file for collecting references
/usr/tmp/bibp*	output of pass one of bib

SEE ALSO

A UNIX Bibliographic Database Facility, Timothy A. Budd and Gary M. Levin, University of Arizona Technical Report 82-1, 1982. (includes format designers guide).
invert(1), troff(1)

NAME

biff − be notified if mail arrives and who it is from

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. When mail notification is enabled, the header and first few lines of the message will be 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), comsat(8C)

NAME

binmail − 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*.

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, it reads a line from the standard input to direct disposition of the message.

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.

An interrupt normally terminates the *mail* command; the mail file is unchanged. The optional argument **−i** tells *mail* to continue after interrupts.

When *persons* are named, *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). To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)).

The **−f** option causes the named file, for example, 'mbox', to be printed as if it were the mail file.

When a user logs in he is informed of the presence of mail.

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

Mail(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 *mail* a set-user-id command that owns the mail directory.

NAME

cal − print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

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

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.
Beware that 'cal 78' refers to the early Christian era, not the 20th century.

NAME

calendar − reminder service

SYNOPSIS

calendar [−]

DESCRIPTION

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

When an argument is present, *calendar* does its job for every user who has a file 'calendar' in his 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 will usually be shared by all users, maintained and documented by the local administration.

FILES

calendar
/usr/lib/calendar to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
/lib/cpp, egrep, sed, mail as subprocesses

SEE ALSO

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

BUGS

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

NAME

cat − catenate and print

SYNOPSIS

cat [**−u**] [**−n**] [**−s**] [**−v**] file ...

DESCRIPTION

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

```
cat file
```

displays the file on the standard output, and

```
cat file1 file2 >file3
```

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

If no input file is given, or if the argument '−' is encountered, *cat* reads from the standard input file. Output is buffered in the block size recommended by *stat*(2) unless the standard output is a terminal, when it is line buffered. The **−u** option makes the output completely unbuffered.

The **−n** option displays the output lines preceded by lines numbers, numbered sequentially from 1. Specifying the **−b** option with the **−n** option omits the line numbers from blank lines.

The **−s** option crushes out multiple adjacent empty lines so that the output is displayed single spaced.

The **−v** option 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 printed as M- (for meta) followed by the character of the low 7 bits. A **−e** option may be given with the **−v** option, which displays a '$' character at the end of each line. Specifying the **−t** option with the **−v** option 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 [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler. *Cc* accepts several types of arguments:

Arguments 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 normally deleted, however, if a single C program is compiled and loaded all at one go.

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

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

−c Suppress the loading 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 for the obsolete debugger *sdb*(1). Also pass the **−lg** flag to *ld*(1).

−w Suppress warning diagnostics.

−p Arrange for the compiler to produce code which counts the number of times each routine is called. If loading takes place, replace the standard startup routine by one which automatically calls *monitor*(3) at the start and arranges to write out a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

−pg Causes the compiler to produce counting 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. Also, a profiling library is searched, in lieu of the standard C library. An execution profile can then be generated by use of *gprof*(1).

−O Invoke an object-code improver.

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

−S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s'.

−M Run only the macro preprocessor on the named C programs, requesting it to generate Makefile dependencies and send the result to the standard output.

−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 eliding comments.

−o *output*
Name the final output file *output*. If this option is used the file 'a.out' will be left undisturbed.

−D*name=def*
−D*name* Define the *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 directories on a standard list.

–L*dir* Library archives are sought first in directories named in **–L** options, then in directories on a standard list.

–f Use an alternate compiler which does not convert expressions involving only floats to double. This does not conform to the standard which states that all intermediate results should be converted to double but does provide a speed improvement for programs which don't require full double precision. This option also makes **register float** variables work appropriately.

–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/c/'.

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. 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.**

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/lib/cpp	preprocessor
/lib/ccom	compiler
/lib/sccom	compiler for single precision floats
/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/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
B. W. Kernighan, *Programming in C—a tutorial*
D. M. Ritchie, *C Reference Manual*
monitor(3), prof(1), gprof(1), adb(1), ld(1), dbx(1), as(1)

DIAGNOSTICS

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

BUGS

The compiler currently ignores advice to put **char**, **unsigned char**, **short**, **unsigned short**, **float**, or **double** variables in registers, except as noted above. It previously produced poor, and in some cases incorrect, code for such declarations.

NAME

cd − change working directory

SYNOPSIS

cd directory

DESCRIPTION

Directory becomes the new working directory. The process must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command. It is therefore recognized and executed by the shells. 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

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. Delimeters 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.

Additional pairs of macros can be added to the list using the **−a** option. This 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

The **−c** option defines commands which would otherwise be complained about as undefined.

The **−f** option requests *checknr* to ignore \f font changes.

The **−s** option requests *checknr* to ignore \s size changes.

Checknr is intended to be used on documents that are prepared with *checknr* in mind, much the same as *lint.* 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.

SEE ALSO

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

DIAGNOSTICS

Complaints about unmatched delimiters.
Complaints about unrecognized commands.
Various complaints about the syntax of commands.

BUGS

There is no way to define a 1 character macro name using **−a**.
Does not correctly recognize certain reasonable constructs, such as conditionals.

NAME

chfn, chsh, passwd − change password file information

SYNOPSIS

passwd [**−f**] [**−s**] [name]

DESCRIPTION

This command changes (or installs) a password, login shell (**−s** option), or GECOS information field (**−f** option) associated with the user *name* (your own name by default).

When altering a password, the program prompts for the current 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.

When altering a login shell, *passwd* displays the current login shell and then prompts for the new one. The new login shell must be one of the approved shells listed in */etc/shells* unless you are the super-user. If */etc/shells* does not exist, the only shells that may be specified are */bin/sh* and */bin/csh*.

The super-user may change anyone's login shell; normal users may only change their own login shell.

When altering the GECOS information field, *passwd* displays the current information, broken into fields, as interpreted by the *finger*(1) program, among others, and prompts for new values. These fields include a user's ''real life'' name, office room number, office phone number, and home phone number. Included in each prompt is a default value, which is enclosed between brackets. The default value is accepted simply by typing a carriage return. To enter a blank field, the word ''none'' may be typed. Below is a sample run:

> **Name [Biff Studsworth II]:**
> **Room number (Exs: 597E or 197C) []:** 521E
> **Office Phone (Ex: 1632) []:** 1863
> **Home Phone (Ex: 987532) [5771546]:** none

Passwd allows phone numbers to be entered with or without hyphens. It is a good idea to run *finger* after changing the GECOS information to make sure everything is setup properly.

The super-user may change anyone's GECOS information; normal users may only change their own.

FILES

/etc/passwd	The file containing all of this information
/etc/shells	The list of approved shells

SEE ALSO

login(1), finger(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, *UNIX password security*

NAME

chgrp − change group

SYNOPSIS

chgrp [-f -R] 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.

When the **−R** option is given, *chgrp* recursively descends its directory arguments setting the specified group-ID. When symbolic links are encountered, their group is changed, but they are not traversed.

FILES

/etc/group

SEE ALSO

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

NAME

chmod − change mode

SYNOPSIS

chmod [**−Rf**] 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 will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **X** (set execute only if file is a directory or some other execute bit is set), **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.

When the **−R** option is given, *chmod* recursively descends its directory arguments setting the mode for each file as described above. When symbolic links are encountered, their mode is not changed and they are not traversed.

If the **−f** option is given, *chmod* will not complain if it fails to change the mode on a file.

EXAMPLES

The first example denies write permission to others, the second makes a file executable by all if it is executable by anyone:

```
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

chfn, chsh, passwd − change password file information

SYNOPSIS

passwd [**−f**] [**−s**] [name]

DESCRIPTION

This command changes (or installs) a password, login shell (**−s** option), or GECOS information field (**−f** option) associated with the user *name* (your own name by default).

When altering a password, the program prompts for the current 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.

When altering a login shell, *passwd* displays the current login shell and then prompts for the new one. The new login shell must be one of the approved shells listed in */etc/shells* unless you are the super-user. If */etc/shells* does not exist, the only shells that may be specified are */bin/sh* and */bin/csh*.

The super-user may change anyone's login shell; normal users may only change their own login shell.

When altering the GECOS information field, *passwd* displays the current information, broken into fields, as interpreted by the *finger*(1) program, among others, and prompts for new values. These fields include a user's ''real life'' name, office room number, office phone number, and home phone number. Included in each prompt is a default value, which is enclosed between brackets. The default value is accepted simply by typing a carriage return. To enter a blank field, the word ''none'' may be typed. Below is a sample run:

> **Name [Biff Studsworth II]:**
> **Room number (Exs: 597E or 197C) []:** 521E
> **Office Phone (Ex: 1632) []:** 1863
> **Home Phone (Ex: 987532) [5771546]:** none

Passwd allows phone numbers to be entered with or without hyphens. It is a good idea to run *finger* after changing the GECOS information to make sure everything is setup properly.

The super-user may change anyone's GECOS information; normal users may only change their own.

FILES

/etc/passwd	The file containing all of this information
/etc/shells	The list of approved shells

SEE ALSO

login(1), finger(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, *UNIX password security*

NAME

clear − clear terminal screen

SYNOPSIS

clear

DESCRIPTION

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

FILES

/etc/termcap terminal capability data base

NAME

cmp − compare two files

SYNOPSIS

cmp [**−l**] [**−s**] file1 file2

DESCRIPTION

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

Options:

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

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

SEE ALSO

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 line feeds

SYNOPSIS

col [**−bfh**]

DESCRIPTION

Col reads the standard input and writes the standard output. It performs the 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.

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

If the **−b** option is given, *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.

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.

If the **−h** option is given, *col* converts white space to tabs to shorten printing time.

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.

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.

The optional **−** suppresses all underlining. It is especially useful for previewing *allboxed* tables from *tbl*(1).

The option **−2** causes all half-lines to be printed, effectively double spacing the output. Normally, a minimal space output format is used which will suppress empty lines. The program 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.

A typical use of *colcrt* would be

    tbl exum2.n | nroff −ms | colcrt − | more

SEE ALSO

nroff/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 file. Input is taken from standard input. Output is sent to standard output.

If called with one parameter the columns of each line will be removed starting with the specified column. If called with two parameters the columns from the first column to the last column will be removed.

Column numbering starts with column 1.

SEE ALSO

expand(1)

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** is a no-op.

SEE ALSO

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

NAME

compress, uncompress, zcat − compress and expand data

SYNOPSIS

compress [**−f**] [**−v**] [**−c**] [**−b** *bits*] [*name ...*]
uncompress [**−f**] [**−v**] [**−c**] [*name ...*]
zcat [*name ...*]

DESCRIPTION

Compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with the extension **.Z,** while keeping the same ownership modes, access and modification times. If no files are specified, the standard input is compressed to the standard output. Compressed files can be restored to their original form using *uncompress* or *zcat.*

The **−f** option will force compression of *name*, even if it does not actually shrink or the corresponding *name*.Z file already exists. Except when run in the background under */bin/sh*, if **−f** is not given the user is prompted as to whether an existing *name*.Z file should be overwritten.

The **−c** (''cat'') option makes *compress/uncompress* write to the standard output; no files are changed. The nondestructive behavior of *zcat* is identical to that of *uncompress* **−c.**

Compress uses the modified Lempel-Ziv algorithm popularized in "A Technique for High Performance Data Compression", Terry A. Welch, *IEEE Computer,* vol. 17, no. 6 (June 1984), pp. 8-19. Common substrings in the file are first replaced by 9-bit codes 257 and up. When code 512 is reached, the algorithm switches to 10-bit codes and continues to use more bits until the limit specified by the **−b** flag is reached (default 16). *Bits* must be between 9 and 16. The default can be changed in the source to allow *compress* to be run on a smaller machine.

After the *bits* limit is attained, *compress* periodically checks the compression ratio. If it is increasing, *compress* continues to use the existing code dictionary. However, if the compression ratio decreases, *compress* discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next "block" of the file.

Note that the **−b** flag is omitted for *uncompress,* since the *bits* parameter specified during compression is encoded within the output, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is attempted.

The amount of compression obtained depends on the size of the input, the number of *bits* per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50−60%. Compression is generally much better than that achieved by Huffman coding (as used in *pack*), or adaptive Huffman coding (*compact*), and takes less time to compute.

The **−v** option causes the printing of the percentage reduction of each file.

If an error occurs, exit status is 1, else if the last file was not compressed because it became larger, the status is 2; else the status is 0.

DIAGNOSTICS

Usage: compress [−fvc] [−b maxbits] [file ...]
    Invalid options were specified on the command line.

Missing maxbits
    Maxbits must follow **−b**.

file: not in compressed format
    The file specified to *uncompress* has not been compressed.

file: compressed with *xx* bits, can only handle *yy* bits
    File was compressed by a program that could deal with more *bits* than the compress code on this machine. Recompress the file with smaller *bits*.

file: already has .Z suffix -- no change
    The file is assumed to be already compressed. Rename the file and try again.

file: filename too long to tack on .Z

The file cannot be compressed because its name is longer than 12 characters. Rename and try again. This message does not occur on BSD systems.

file already exists; do you wish to overwrite (y or n)?
: Respond "y" if you want the output file to be replaced; "n" if not.

uncompress: corrupt input
: A SIGSEGV violation was detected which usually means that the input file is corrupted.

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

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

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

-- file unchanged
: No savings is achieved by compression. The input remains virgin.

BUGS

Although compressed files are compatible between machines with large memory, **−b**12 should be used for file transfer to architectures with a small process data space (64KB or less, as exhibited by the DEC PDP series, the Intel 80286, etc.)

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

NAME

cp − copy

SYNOPSIS

cp [**−ip**] file1 file2

cp [**−ipr**] file ... directory

DESCRIPTION

File1 is copied onto *file2*. By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the current *umask*(2) is used. The **−p** option causes *cp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the present *umask*.

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.

If the **−i** option is specified, *cp* will prompt the user with the name of the file whenever the copy will cause an old file to be overwritten. An answer of 'y' will cause *cp* to continue. Any other answer will prevent it from overwriting the file.

If the **−r** option is specified and any of the source files are directories, *cp* copies each subtree rooted at that name; in this case the destination must be a directory.

SEE ALSO

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

NAME

crypt − encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

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

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

will print the clear.

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

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

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

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

BUGS

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, Bell Telephone Laboratories assumes no responsibility for their use by the recipient. Further, Bell Laboratories assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

NAME

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

SYNOPSIS

csh [**−cefinstvVxX**] [arg ...]

DESCRIPTION

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

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

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

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

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

Lexical structure

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

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

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

Commands

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

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

Jobs

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

[1] 1234

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

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

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

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

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

Status reporting

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

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

File Name Completion

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

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

and the input is

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

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

% vi chaosnet

However, given

% vi D<escape>

csh will only expand the input to

% vi DSC.

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

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

% vi D<control-D>

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

DSC.NEW DSC.OLD

while the input line remains unchanged.

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

cd ~ro<control-D>

may produce the expansion

cd ~root

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

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

% set fignore = (.o .out)

then typing

% vi x<escape>

would result in the completion to

% vi xmpl.c

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

Substitutions

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

History substitutions

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

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

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

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

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

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

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

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

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

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.xxx' component, leaving the root name.
e	Remove all but the extension '.xxx' part.
s/*l*/*r*/	Substitute *l* for *r*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. 'g&'.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

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

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

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

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

Quotations with ´ and "

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

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

Alias substitution

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

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

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

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

Variable substitution

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

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

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

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

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

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

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

$name
${name}
: Are replaced by the words of the value of variable *name,* each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.
If *name* is not a shell variable, but is set in the environment, then that value is returned (but **:** 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 ‘~’, then that word is a candidate for filename substitution, also known as ‘globbing’. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters ‘*’, ‘?’ and ‘[’ imply pattern matching, the characters ‘~’ and ‘{’ being more akin to abbreviations.

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

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

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

Input/output

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

< name

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

<< word

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

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

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

If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a 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, then it is an error for the file not to exist unless one of the ‘!’ forms is given. Otherwise similar to ‘>’.

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

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

Expressions

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

|| && | ↑ & == != =~ !~ <= >= < > << >> + − * / % ! ~ ( )

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

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

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

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

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

Control flow

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

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

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

Builtin commands

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

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

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

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

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

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

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

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

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

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

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

echo wordlist
echo −n wordlist
The specified words are written to the shells standard output, separated by spaces, and terminated with a 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 in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of using *eval*.

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

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

fg

fg %job ...

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

foreach name (wordlist)
...
end

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

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are 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, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

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

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

jobs
jobs −l

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

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

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

limit
limit *resource*
limit *resource maximum-use*
limit −h
limit −h *resource*
limit −h *resource maximum-use*

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

Resources controllable currently include *cputime* (the maximum number of cpu-seconds to be used by each 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 scheduling priority for this shell to 4. The second form sets the priority to the given number. The final two forms run command at priority 4 and *number* respectively. The greater the number, the less cpu the process will get. The super-user may specify negative priority 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 which is to terminate shell scripts or to return to the terminal command input level. The second form 'onintr −' causes all interrupts to be ignored. The final form causes the shell to execute a 'goto label' when an interrupt is received or a child process terminates because it was interrupted.

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

popd
popd +n
Pops the directory stack, returning to the new top directory. With an argument '+*n*' discards the *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 (ala *cd)* and pushes the old current working directory (as in *csw)* onto the directory stack. With a numeric argument, rotates the *n* th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

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

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

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

of words in *wordlist.* In all cases the value is command and filename expanded.

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

setenv
setenv name value
setenv name

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

shift
shift variable

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

source name
source −h name

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

stop
stop %job ...

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

suspend

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

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

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

time
time command

With no argument, a summary of time used by this 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
unlimit *resource*
unlimit –h
unlimit –h *resource*

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

unset pattern

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

unsetenv pattern

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

wait

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

while (expr)
...
end

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

%job

Brings the specified job into the foreground.

%job **&**

Continues the specified job in the background.

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

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

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

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

Pre-defined and environment variables

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

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

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

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

cwd The full pathname of the current directory.

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

filec Enable file name completion.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Non-builtin command execution

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

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

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

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

Argument list processing

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

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

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

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

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

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

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

−s Command input is taken from the standard input.

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

−v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

−x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

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

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

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

Signal handling

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

AUTHOR

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

FILES

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

LIMITATIONS

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

SEE ALSO

sh(1), access(2), execve(2), fork(2), killpg(2), pipe(2), sigvec(2), umask(2), setrlimit(2), wait(2), tty(4), a.out(5), environ(7), 'An introduction to the C shell'

BUGS

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

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

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

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

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

It should be possible to use the ':' modifiers on the output of command substitutions. All and more than one ':' modifier should be allowed on '$' substitutions.

The way the **filec** facility is implemented is ugly and expensive.

NAME

ctags − create a tags file

SYNOPSIS

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

DESCRIPTION

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

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

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

```
ctags −v files | sort −f > index
vgrind −x index
```

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

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

Other options are:

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

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

−a append to tags file.

−t create tags for typedefs.

−w suppressing warning diagnostics.

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

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

FILES

tags output tags file

SEE ALSO

ex(1), vi(1)

AUTHOR

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

BUGS

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

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

Does not know about #ifdefs.

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

NAME

date − print and set the date

SYNOPSIS

date [-n] [-u] [yymmddhhmm [**.**ss]]

DESCRIPTION

If no arguments are given, the current date and time are printed. Providing an argument will set the desired date. Only the superuser can set the date. The *-u* flag is used to display or set the date in GMT (universal) time. *yy* represents the last two digits of the year; the first *mm* is the month number; *dd* is the day number; *hh* is the hour number (24 hour system); the second *mm* is the minute number; ***.**ss* is optional and represents the seconds. For example:

date 8506131627

sets the date to June 13 1985, 4:27 PM. The year, month and day may be omitted; the default values will be the current ones. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight-saving time.

If *timed(8)* is running to synchronize the clocks of machines in a local area network, *date* sets the time globally on all those machines unless the **−n** option is given.

FILES

/usr/adm/wtmp to record time-setting. In /usr/adm/messages, *date* records the name of the user setting the time.

SEE ALSO

gettimeofday(2), utmp(5), timed(8),
TSP: The Time Synchronization Protocol for UNIX 4.3BSD, R. Gusella and S. Zatti

DIAGNOSTICS

Exit status is 0 on success, 1 on complete failure to set the date, and 2 on successfully setting the local date but failing globally.

'You are not superuser: date not set' if you try to change the date but are not the super-user. Occasionally, when *timed* synchronizes the time on many hosts, the setting of a new time value may require more than a few seconds. On these occasions, *date* prints: 'Network time being set'. The message 'Communication error with timed' occurs when the communication between *date* and *timed* fails.

BUGS

The system attempts to keep the date in a format closely compatible with VMS. VMS, however, uses local time (rather than GMT) and does not understand daylight-saving time. Thus, if you use both UNIX and VMS, VMS will be running on GMT.

NAME

dbx − debugger

SYNOPSIS

dbx [**−r**] [**−i**] [**−k**] [**−I** *dir*] [**−c** *file*] [*objfile* [*coredump*]]

DESCRIPTION

Dbx is a tool for source level debugging and execution of programs under UNIX. The *objfile* is an object file produced by a compiler with the appropriate flag (usually ‘‘−g’’) specified to produce symbol information in the object file. Currently, *cc*(1), *f77*(1), *pc*(1), and the DEC Western Research Laboratory Modula-2 compiler, *mod*(l), produce the appropriate source information. The machine level facilities of *dbx* can be used on any program.

The object file contains a symbol table that includes the name of the 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.

If the file ‘‘.dbxinit’’ exists in the current directory then the debugger commands in it are executed. *Dbx* also checks for a ‘‘.dbxinit’’ in the user's home directory if there isn't one in the current directory.

The command line options and their meanings are:

−r Execute *objfile* immediately. If it terminates successfully *dbx* exits. Otherwise the reason for termination will be reported and the user offered the option of entering the debugger or letting the program fault. *Dbx* will read from ‘‘/dev/tty’’ when **−r** is specified and standard input is not a terminal.

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

−k Map memory addresses, useful for kernel debugging.

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

−c *file* Execute the *dbx* commands in the *file* before reading from standard input.

Unless **−r** is specified, *dbx* just prompts and waits for a command.

Execution and Tracing Commands

run [*args*] [**<** *filename*] [**>** *filename*]
rerun [*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. When **rerun** is used without any arguments the previous argument list is passed to the program; otherwise it is identical to **run**. If *objfile* has been written since the last time the symbolic information was read in, *dbx* will read in the new information.

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

Have tracing information printed when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

The first argument describes what is to be traced. If it is a *source-line-number*, then the line is printed immediately prior to being executed. 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, e.g. "mumble.p":17.

If the argument is a procedure or function name then every time it is called, information is printed 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 then the value it is returning is also printed.

If the argument is an *expression* with an **at** clause then the value of the expression is printed whenever the identified source line is reached.

If the argument is a variable then the name and value of the variable is printed whenever it changes. Execution is substantially slower during this form of tracing.

If no argument is specified then all source lines are printed before they are executed. Execution is substantially slower during this form of tracing.

The clause ''**in** *procedure/function*'' restricts tracing information to be printed only while executing inside the given procedure or function.

Condition is a boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

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.

status [**>** *filename*]
Print out the currently active **trace** and **stop** commands.

delete *command-number ...*
The traces or stops corresponding to the given numbers are removed. The numbers associated with traces and stops are printed by the **status** command.

catch *number*
catch *signal-name*
ignore *number*
ignore *signal-name*
Start or stop trapping a signal before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. A signal may be specified by number or by a name (e.g., SIGINT). Signal names are case insensitive and the ''SIG'' prefix is optional. By default all signals are trapped except SIGCONT, SIGCHILD, SIGALRM and SIGKILL.

cont *integer*
cont *signal-name*
Continue execution from where it stopped. If a signal is specified, the process continues as though it received the signal. Otherwise, the process is continued as though it had not been stopped.

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 to examine the program state.

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 the **step** command will stop at the beginning of that block, while the **next** command will not.

return [*procedure*]
Continue until a return to *procedure* is executed, or until the current procedure returns if none is specified.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function.

Printing Variables and Expressions

Names are resolved first using the static scope of the current function, then using the dynamic scope if the name is not defined in the static scope. If static and dynamic searches do not yield a result, an arbitrary symbol is chosen and the message ''[using *qualified name*]'' is printed. The name resolution procedure may be overridden by qualifying an identifier with a block name, e.g., ''*module.variable*''. For C, source files are treated as modules named by the file name without ''.c''.

Expressions are specified with an approximately common subset of C and Pascal (or equivalently Modula-2) syntax. Indirection can be denoted using either a prefix ''*'' or a postfix ''^'' and array expressions are subscripted by brackets (''[ ]''). The field reference operator (''.'') can be used with pointers as well as records, making the C operator ''->'' unnecessary (although it is supported).

Types of expressions are checked; the type of an expression may be overridden by using ''*type-name*(*expression*)''. When there is no corresponding named type the special constructs ''&*type-name*'' and ''$$*tag-name*'' can be used to represent a pointer to a named type or C structure tag.

assign *variable* **=** *expression*
Assign the value of the expression to the variable.

dump [*procedure*] [**>** *filename*]
Print the names and values of variables in the given procedure, or the current one if none is specified. If the procedure given is ''.'', then the all active variables are dumped.

print *expression* [**,** *expression* ...]
Print out the values of the expressions.

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, i.e. the outer blocks that the identifier is associated with.

up [*count*]
down [*count*]
Move the current function, which is used for resolving names, up or down the stack *count* levels. The default *count* is 1.

where Print out a list of the active procedures and function.

whereis *identifier*
Print the full qualification of all the symbols whose name matches the given identifier. The order in which the symbols are printed is not meaningful.

Accessing Source Files

/regular expression[/]
?regular expression[?]
: Search forward or backward in the current source file for the given pattern.

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*. If none is specified then the current source file name is printed.

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

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.

use *directory-list*
: Set the list of directories to be searched when looking for source files.

Command Aliases and Variables

alias *name name*
alias *name* ''*string*''
alias *name* (*parameters*) ''*string*''
: When commands are processed, dbx first checks to see if the word is an alias for either a command or a string. If it is an alias, then dbx treats the input as though the corresponding string (with values substituted for any parameters) had been entered. For example, to define an alias ''rr'' for the command ''rerun'', one can say

```
alias rr rerun
```

To define an alias called ''b'' that sets a stop at a particular line one can say

```
alias b(x) ''stop at x''
```

Subsequently, the command ''b(12)'' will expand to ''stop at 12''.

set *name* [= *expression*]
: The **set** command defines values for debugger variables. The names of these variables cannot conflict with names in the program being debugged, and are expanded to the corresponding expression within other commands. The following variables have a special meaning:

$frame

Setting this variable to an address causes dbx to use the stack frame pointed to by the address for doing stack traces and accessing local variables. This facility is of particular use for kernel debugging.

$hexchars
$hexints
$hexoffsets
$hexstrings
When set, dbx prints out out characters, integers, offsets from registers, or character pointers respectively in hexadecimal.

$listwindow
The value of this variable specifies the number of lines to list around a function or when the **list** command is given without any parameters. Its default value is 10.

$mapaddrs
Setting (unsetting) this variable causes dbx to start (stop) mapping addresses. As with ''$frame'', this is useful for kernel debugging.

$unsafecall
$unsafeassign
When ''$unsafecall'' is set, strict type checking is turned off for arguments to subroutine or function calls (*e.g.* in the **call** statement). When ''$unsafeassign'' is set, strict type checking between the two sides of an **assign** statement is turned off. These variables should be used only with great care, because they severely limit dbx's usefulness for detecting errors.

unalias *name*
Remove the alias with the given name.

unset *name*
Delete the debugger variable associated with *name*.

Machine Level Commands

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*address*] [**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*]
Print the contents of memory starting at the first *address* and continuing up to the second *address* or until *count* items are printed. If the address is ''.'', the address following the one printed most recently is used. The *mode* specifies how memory is to be printed; if it is omitted the previous mode specified is used. The initial mode is ''X''. The following modes are supported:

i print the machine instruction
d print a short word in decimal
D print a long word in decimal

o	print a short word in octal
O	print a long word in octal
x	print a short word in hexadecimal
X	print a long word in hexadecimal
b	print a byte in octal
c	print a byte as a character
s	print a string of characters terminated by a null byte
f	print a single precision real number
g	print a double precision real number

Symbolic addresses are specified by preceding the name with an ‘‘&’’. Registers are denoted by ‘‘$rN’’ where N is the number of the register. Addresses may be expressions made up of other addresses and the operators ‘‘+’’, ‘‘-’’, and indirection (unary ‘‘*’’).

Miscellaneous Commands

gripe Invoke a mail program to send a message to the person in charge of *dbx*.

help Print out a synopsis of *dbx* commands.

quit Exit *dbx*.

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

source *filename*
Read *dbx* commands from the given *filename*.

FILES

a.out	object file
.dbxinit	initial commands

SEE ALSO

cc(1), f77(1), pc(1), mod(l)

COMMENTS

Dbx suffers from the same ‘‘multiple include’’ malady as did *sdb*. 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 would not save much time, though it would reduce some of the disk space used.

This 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. However, even with Modula-2 there is a substantial amount of duplication of symbol information necessary for inter-module type checking.

Some problems remain with the support for individual languages. Fortran problems include: inability to assign to logical, logical*2, complex and double complex variables; inability to represent parameter constants which are not type integer or real; peculiar representation for the values of dummy procedures (the value shown for a dummy procedure is actually the first few bytes of the procedure text; to find the location of the procedure, use ‘‘&’’ to take the address of the variable).

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.

An example which prints the first ten values of n! is

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

option	*values*
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs;* also, if no conversion is specified, it is particularly efficient since no 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 where input is a magtape or similar device).
seek=*n*	seek *n* records from beginning of output file before copying
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
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, b** or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii, 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.

For example, to read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x:*

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

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

SEE ALSO

cp(1), 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.
One must specify ''conv=noerror,sync'' when copying raw disks with bad sectors to insure *dd* stays synchronized.

Certain combinations of arguments to *conv*= are permitted. However, the *block* or *unblock* option cannot be combined with *ascii*, *ebcdic* or *ibm*. Invalid combinations *silently ignore* all but the last mutually-exclusive keyword.

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.

If the **−w** flag is given, the output is a word list, one 'word' (string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed) per line, and all other characters ignored. Otherwise, the output follows the original, with the deletions mentioned above.

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 − disk free

SYNOPSIS

df [**−i**] [filesystem ...] [file ...]

DESCRIPTION

Df prints out the amount of free disk space available on the specified *filesystem,* e.g. ''/dev/rp0a'', or on the filesystem in which the specified *file,* e.g. ''$HOME'', is contained. If no file system is specified, the free space on all of the normally mounted file systems is printed. The reported numbers are in kilobytes.

Other options are:

−i Report also the number of inodes which are used and free.

FILES

/etc/fstab list of normally mounted filesystems

SEE ALSO

fstab(5), icheck(8), quot(8)

NAME

diction, explain − print wordy sentences; thesaurus for diction

SYNOPSIS

diction [**−ml**] [**−mm**] [**−n**] [**−f** pfile] file ...
explain

DESCRIPTION

Diction finds all sentences in a document that contain phrases from a data base of bad or wordy diction. Each phrase is bracketed with [ ]. Because *diction* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **−ms** may be overridden with the flag **−mm.** The flag **−ml** which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The user may supply her/his own pattern file to be used in addition to the default file with **−f** *pfile.* If the flag **−n** is also supplied the default file will be suppressed.

Explain is an interactive thesaurus for the phrases found by diction.

SEE ALSO

deroff(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks. In particular, *diction* doesn't grok **−me.**

NAME

diff − differential file and directory comparator

SYNOPSIS

diff [**−l**] [**−r**] [**−s**] [**−cefhn**] [**−biwt**] dir1 dir2
diff [**−cefhn] [−biwt**] file1 file2
diff [**−D***string*] [**−biw**] file1 file2

DESCRIPTION

If both arguments are directories, *diff* sorts the contents of the directories by name, and then runs the regular file *diff* algorithm (described below) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed. Options when comparing directories are:

−l long output format; each text file *diff* is piped through *pr*(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.

−r causes application of *diff* recursively to common subdirectories encountered.

−s causes *diff* to report files which are the same, which are otherwise not mentioned.

−Sname
starts a directory *diff* in the middle beginning with file *name.*

When run on regular files, and when comparing text files which differ during directory comparison, *diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, *diff* finds a smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, then either may be given as '−', in which case the standard input is used. If *file1* is a directory, then a file in that directory whose file-name is the same as the file-name of *file2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

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

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

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

Except for **−b, -w, -i** or **-t** which may be given with any of the others, the following options are mutually exclusive:

−e produces a script of *a, c* and *d* commands for the editor *ed,* which will recreate *file2* from *file1*. In connection with **−e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

(shift; cat $*; echo ´1,$p´) | ed − $1

Extra commands are added to the output when comparing directories with **−e,** so that the result is a *sh*(1) script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2.*

−f produces a script similar to that of **−e,** not useful with *ed,* and in the opposite order.

−n produces a script similar to that of **−e,** but in the opposite order and with a count of changed lines on each insert or delete command. This is the form used by *rcsdiff*(1).

−c produces a diff with lines of context. The default is to present 3 lines of context and may be changed, e.g to 10, by **−c10**. With **−c** the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from *file1* are marked with '– '; those added to *file2* are marked '+ '. Lines which are changed from one file to the other are marked in both files with with '! '.

Changes which lie within <context> lines of each other are grouped together on output. (This is a change from the previous ''diff -c'' but the resulting output is usually much easier to interpret.)

–h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

–Dstring causes *diff* to create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1,* while defining *string* will yield *file2.*

–b causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equal.

–w is similar to **–b** but causes whitespace (blanks and tabs) to be totally ignored. E.g., ''if (a == b)'' will compare equal to ''if(a==b)''.

–i ignores the case of letters. E.g., ''A'' will compare equal to ''a''.

–t will expand tabs in output lines. Normal or **–c** output adds character(s) to the front of each line which may screw up the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

FILES

/tmp/d?????
/usr/lib/diffh for **–h**
/bin/diff for directory diffs
/bin/pr

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff3(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, -w** or **-i** options specified, *diff* first compares the files ala *cmp,* and then decides to run the *diff* algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

NAME

diff3 − 3-way differential file comparison

SYNOPSIS

diff3 [**−exEX3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

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

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

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

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

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

```
(cat script; echo ´1,$p´) | ed − file1
```

The **−E** and **−X** are similar to **−e** and **−x**, respectively, but treat overlapping changes (i.e., changes that would be flagged with ==== in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by "<<<<<<" and ">>>>>>" lines.

For example, suppose lines 7-8 are changed in both file1 and file2. Applying the edit script generated by the command

"diff3 -E file1 file2 file3"

to file1 results in the file:

```
lines 1-6
of file1
<<<<<<< file1
lines 7-8
of file1
=======
lines 7-8
of file3
>>>>>>> file3
rest of file1
```

The **−E** option is used by RCS *merge*(1) to insure that overlapping changes in the merged files are preserved and brought to someone's attention.

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(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, '**.**' is used.

The argument **−s** causes only the grand total to be given. The argument **−a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

A file which has two links to it is only counted once.

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**] [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a newline on the standard output. If the flag **−n** is used, no newline is added to the output.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, do 'echo ... 1>&2'.

NAME

ed − text editor

SYNOPSIS

ed [**−**] [**−x**] [name]

DESCRIPTION

Ed is the standard text editor.

If a *name* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed's* buffer so that it can be edited. If **−x** is present, an *x* command is simulated first to handle an encrypted file. The optional **−** suppresses the printing of explanatory output and should be used when the standard input is an editor script.

Ed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*.

Commands to *ed* have a simple and regular structure: zero or more *addresses* followed by a single character *command,* possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

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

Ed supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus \[**.** and sometimes ^*$.
2. A **.** matches any character.
3. A \ followed by any character except a digit or ( ) matches that character.
4. A nonempty string *s* bracketed [*s*] (or [^*s*]) matches any character in (or not in) *s.* In *s,* \ has no special meaning, and] may only appear as the first letter. A substring *a−b,* with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of form 1-4 followed by * matches a sequence of 0 or more matches of the regular expression.
6. A regular expression, *x,* of form 1-8, bracketed \(*x*\) matches what *x* matches.
7. A \ followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *n*th \(matched.
8. A regular expression of form 1-8, *x,* followed by a regular expression of form 1-7, *y* matches a match for *x* followed by a match for *y,* with the *x* match being as long as possible while still permitting a *y* match.
9. A regular expression of form 1-8 preceded by ^ (or followed by $), is constrained to matches that begin at the left (or end at the right) end of a line.
10. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.
11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) 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.

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

1. The character '**.**' addresses the current line.
2. The character '$' addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. ''*x*' addresses the line marked with the name *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A regular expression enclosed in slashes '/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.
6. A regular expression enclosed in queries '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.
7. An address followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with '+' or '−' the addition or subtraction is taken with respect to the current line; e.g. '−5' is understood to mean '**.**−5'.
9. If an address ends with '+' or '−', then 1 is added (resp. subtracted). As a consequence of this rule and rule 8, the address '−' refers to the line before the current line. Moreover, trailing '+' and '−' characters have cumulative effect, so '−−' refers to the current line less 2.
10. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is equivalent to '−'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma '**,**'. They may also be separated by a semicolon '**;**'. In this case the current line '**.**' is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches ('/', '?'). The second address of any two-address sequence must correspond to a line following the line corresponding to the first address. The special form '%' is an abbreviation for the address pair '1,$'.

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

As mentioned, it is generally illegal for more than one command to appear on a line. However, most commands may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed respectively in the way discussed below. Commands may also be suffixed by 'n', meaning the output of the command is to be line numbered. These suffixes may be combined in any order.

(**.**) a
<text>
.

The append command reads the given text and appends it after the addressed line. '**.**' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.

(**.** , **.**) c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '**.**' is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(**.** , **.**) d

The delete command deletes the addressed lines from the buffer. 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

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in. '**.**' is set to the last line of the buffer. The number of characters read is typed. 'filename' is remembered for possible use as a default file name in a subsequent *r* or *w* command. If 'filename' is missing, the remembered name is used.

E filename

This command is the same as *e,* except that no diagnostic results when no *w* has been given since the last buffer alteration.

f filename

The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1,$) g/regular expression/command list

In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '**.**' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with '\'. *A, i,* and *c* commands and associated input are permitted; the '**.**' terminating input mode may be omitted if it would be on the last line of the command list. The commands *g* and *v* are not permitted in the command list.

(**.**) i

<text>

.

This command inserts the given text before the addressed line. '**.**' is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

(**.** , **.**+1) j

This command joins the addressed lines into a single line; intermediate newlines simply disappear. '**.**' is left at the resulting line.

(**.**) k*x*

The mark command marks the addressed line with name *x,* which must be a lower-case letter. The address form '′*x*' then addresses this line.

(**.** , **.**) l

The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The *l* command may be placed on the same line after any non-i/o command.

(**.** , **.**) m*a*

The move command repositions the addressed lines after the line addressed by *a*. The last of the moved lines becomes the current line.

(. , .) p
The print command prints the addressed lines. '**.**' is left at the last line printed. The *p* command may be placed on the same line after any non-i/o command.

(. , .) P
This command is a synonym for *p.*

q The quit command causes *ed* to exit. No automatic write of a file is done.

Q This command is the same as *q,* except that no diagnostic results when no *w* has been given since the last buffer alteration.

($) r filename
The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands) . The file name is remembered if there was no remembered file name already. Address '0' is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '**.**' is left at the last line read in from the file.

(**.** , **.**) s/regular expression/replacement/ or,
(**.** , **.**) s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, if the global replacement indicator 'g' appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any punctuation character may be used instead of '/' to delimit the regular expression and the replacement. '**.**' is left at the last line substituted.

An ampersand '&' appearing in the replacement 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 '\'.

One or two trailing delimiters may be omitted, implying the 'p' suffix. The special form 's' followed by *no* delimiters repeats the most recent substitute command on the addressed lines. The 's' may be followed by the letters **r** (use the most recent regular expression for the left hand side, instead of the most recent left hand side of a substitute command), **p** (complement the setting of the *p* suffix from the previous substitution), or **g** (complement the setting of the *g* suffix). These letters may be combined in any order.

(. , .) t *a*
This command acts just like the *m* command, except that a copy of the addressed lines is placed after address *a* (which may be 0). '**.**' is left on the last line of the copy.

(. , .) u
The undo command restores the buffer to it's state before the most recent buffer modifying command. The current line is also restored. Buffer modifying commands are *a, c, d, g, i, k,* and *v.* For purposes of undo, *g* and *v* are considered to be a single buffer modifying command. Undo is its own inverse.

When *ed* runs out of memory (at about 8000 lines on any 16 bit mini-computer such as the PDP-11) This full undo is not possible, and *u* can only undo the effect of the most recent substitute on the current line. This restricted undo also applies to editor scripts when *ed* is invoked with the **-** option.

(1, $) v/regular expression/command list
This command is the same as the global command *g* except that the command list is executed *g* with '**.**' initially set to every line *except* those matching the regular expression.

(1, $) w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created. The file name is remembered if there was no remembered file name already. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). '**.**' is unchanged. If the command is successful, the number of characters written is printed.

(1, $) W filename
This command is the same as *w,* except that the addressed lines are appended to the file.

(1, $) wq filename
This command is the same as *w* except that afterwards a *q* command is done, exiting the editor after the file is written.

x A key string is demanded from the standard input. Later *r, e* and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption. (**.**+1) z or,

(**.**+1) z*n*
This command scrolls through the buffer starting at the addressed line. 22 (or *n,* if given) lines are printed. The last line printed becomes the current line. The value *n* is sticky, in that it becomes the default for future *z* commands.

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

!<shell command>
The remainder of the line after the '!' is sent to *sh*(1) to be interpreted as a command. '**.**' is unchanged.

(**.**+1, **.**+1) <newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '.+1p'; it is useful for stepping through text. If two addresses are present with no intervening semicolon, *ed* prints the range of lines. If they are separated by a semicolon, the second line is printed.

If an interrupt signal (ASCII DEL) is sent, *ed* prints '?interrupted' and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and, on mini computers, 128K characters in the temporary file. The limit on the number of lines depends on the amount of core: each line takes 2 words.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.

FILES

/tmp/e*
edhup: work is saved here if terminal hangs up

SEE ALSO

B. W. Kernighan, *A Tutorial Introduction to the ED Text Editor*
B. W. Kernighan, *Advanced editing on UNIX*
ex(1), sed(1), crypt(1)

DIAGNOSTICS

'?name' for inaccessible file; '?self-explanatory message' for other errors.

To protect against throwing away valuable work, a *q* or *e* command is considered to be in error, unless a *w* has occurred since the last buffer change. A second *q* or *e* will be obeyed regardless.

BUGS

The *l* command mishandles DEL.
The *undo* command causes marks to be lost on affected lines.
The *x* command, **-x** option, and special treatment of hangups only work on UNIX.

NAME

efl − Extended Fortran Language

SYNOPSIS

efl [option ...] [filename ...]

DESCRIPTION

Efl compiles a program written in the EFL language into clean Fortran. *Efl* provides the same control flow constructs as does *ratfor*(1), which are essentially identical to those in C:

statement grouping with braces;
: decision-making with if, if-else, and switch-case; while, for, Fortran do, repeat, and repeat...until loops; multi-level break and next. In addition, EFL has C-like data structures, and more uniform and convenient input/output syntax, generic functions. EFL also provides some syntactic sugar to make programs easier to read and write:

free form input:
: multiple statements/line; automatic continuation statement label names (not just numbers),

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

The Efl command option **−w** suppresses warning messages. The option **−C** causes comments to be copied through to the Fortran output (default); **−#** prevents comments from being copied through. If a command argument contains an embedded equal sign, that argument is treated as if it had appeared in an **option** statement at the beginning of the program. *Efl* is best used with *f77*(1).

SEE ALSO

f77(1), ratfor(1).
S. I. Feldman, *The Programming Language EFL*, Bell Labs Computing Science Technical Report #78.

NAME

eqn, neqn, checkeq − typeset mathematics

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

Subscripts and superscripts are produced with the keywords **sub** and **sup.** 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: $\lim\limits_{n \to \infty} \sum\limits_0^n x_i$ is made with *lim from {n−> inf } sum from 0 to n x sub i.*

Left and right brackets, braces, etc., of the right height are made with **left** and **right:** *left [x sup 2 + y sup 2 over alpha right] ~=~1* produces $\left[x^2 + \frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$.

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

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~=~ n under* is $\bar{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \overleftrightarrow{y}$.

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

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

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

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

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

SEE ALSO

troff(1), tbl(1), ms(7), eqnchar(7)
B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics—User's Guide*
J. F. Ossanna, *NROFF/TROFF User's Manual*

BUGS

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

NAME

error − analyze and disperse compiler error messages

SYNOPSIS

error [**−n**] [**−s**] [**−q**] [**−v**] [**−t** suffixlist] [**−I** ignorefile] [name]

DESCRIPTION

Error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

Error looks at the error messages, either from the specified file *name* or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceding to which the line the error message refers. 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. By specifying the **−q** query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. Otherwise *error* proceeds on its merry business. If the **−t** touch option and associated suffix list is given, *error* will restrict itself to touch only those files with suffices in the suffix list. Error also can be asked (by specifying **−v)** to invoke *vi*(1) on the files in which error messages were inserted; this obviates the need to remember the names of the files with errors.

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,

```
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, f77,* and *DEC Western Research Modula-2. Error* knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. For all languages except *Pascal,* error messages are restricted to be on one line. Some error messages refer to more than one line in more than one files; *error* will duplicate the error message and insert it at all of the places referenced.

Error will do one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file it is 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 users'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 will not be inserted into any source file.

file specific Error message 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 candidates for inserting into the file they refer to. 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 preceding the line the language processor found in error. 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, programs with comments and source on the same line should be formatted so that language statements appear before comments.

Options available with *error* are:

−n Do *not* touch any files; all error messages are sent to the standard output.

−q The user is *queried* whether s/he wants to touch the file. A ''y'' or ''n'' to the question is necessary to continue. Absence of the **−q** option implies that all referenced files (except those referring 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 suffixes do not appear in the suffix list are not touched. The suffix list is dot separated, and ''*'' wildcards work. Thus the suffix list:

".c.y.foo*.h"

allows *error* to touch files ending with ''.c'', ''.y'', ''.foo*'' 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.

AUTHOR

Robert Henry

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

If you have not used *ed,* or are a casual user, you will find that the editor *edit* is convenient for you. It avoids some of the complexities of *ex* used mostly by systems programmers and persons very familiar with *ed.*

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

DOCUMENTATION

The document *Edit: A tutorial* (USD:14) provides a comprehensive introduction to *edit* assuming no previous knowledge of computers or the UNIX system.

The *Ex Reference Manual − Version 3.7* (USD:16) 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 (USD:15) introduces the display editor *vi* and provides reference material on *vi.* 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(7)

AUTHOR

Originally written by William Joy
Mark Horton has maintained the editor since version 2.7, adding macros, support for many unusual terminals, and other features such as word abbreviation mode.

BUGS

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

Undo never clears the buffer modified condition.

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

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

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

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

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

NAME

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

SYNOPSIS

expand [−tabstop] [−tab1,tab2,...,tabn] [file ...]
unexpand [**−a**] [file ...]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into 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.

If a single *tabstop* argument is given, then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default, only leading blanks and tabs are reconverted to maximal strings of tabs. If the **−a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

NAME

expr − evaluate arguments as an expression

SYNOPSIS

expr arg **. . .**

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

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
where *relop is one of* < <= = != >= >, yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expr* are integers, otherwise lexicographic.

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*
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).

(*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.

NAME

f77 − Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the UNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.F' are also taken to be Fortran 77 source programs; these are first processed by the C preprocessor before being compiled by *f77.*

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs respectively; these are first transformed by the appropriate preprocessor, then compiled by f77.

Arguments whose names end with '.c' or '.s' are taken to be C or assembly source programs and are compiled or assembled, producing a '.o' file.

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) and pass the **−lg** flag to *ld*(1) so that on abnormal terminations, the memory image is written to file *core*. Incompatible with **−O**.

−o output
Name the final output file *output* instead of 'a.out'.

−p Prepare object files for profiling, see *prof*(1).

−pg Causes the compiler to produce counting 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. An execution profile can then be generated by use of *gprof*(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).

−O Invoke an object-code optimizer. Incompatible with **−g**.

−S Compile the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

The following options are peculiar to *f77*.

−d Used for debugging the compiler.

−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.

−q Suppress printing of file names and program unit names during compilation.

−m Apply the M4 preprocessor to each '.r' file before transforming it with the Ratfor or EFL preprocessor.

−onetrip

−1 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.)

−r8 Treat all floating point variables, constants, functions and intrinsics as double precision and all complex quantities as double complex.

−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. For multi-dimensional arrays, only the equivalent linear subscript is checked.

−F Apply the C preprocessor to '.F' files, and the EFL, or Ratfor preprocessors to '.e' and '.r' files, put the result in the file with the suffix changed to '.f', but do not compile.

−E*x* Use the string *x* as an EFL option in processing '.e' files.

−R*x* Use the string *x* as a Ratfor option in processing '.r' files.

−N[qxscn]*nnn*
Make static tables in the compiler bigger. The compiler will complain if it overflows its tables and suggest 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 (e.g. 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 with name 'a.out'.

Programs compiled with *f77* produce memory dumps in file *core* upon abnormal termination if the **−g** flag was specified during loading. If the environment variable *f77_dump_flag* is set to a value beginning with **y** or **n**, dumps for abnormal terminations are respectively forced or suppressed.

FILES

file.[fFresc]	input file
file.o	object file
a.out	loaded output
/usr/lib/f77pass1	compiler
/lib/f1	pass 2
/lib/c2	optional optimizer
/lib/cpp	C preprocessor
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/usr/lib/libU77.a	UNIX interface library
/usr/lib/libm.a	math library
/lib/libc.a	C library, see section 3
/usr/lib/libF77_p.a	profiling intrinsic function library
/usr/lib/libI77_p.a	profiling Fortran I/O library

/usr/lib/libU77_p.a	profiling UNIX interface library
/usr/lib/libm_p.a	profiling math library
/usr/lib/libc_p.a	profiling 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

S. I. Feldman, P. J. Weinberger, J. Berkman, *A Portable Fortran 77 Compiler*
D. L. Wasley, J. Berkman, *Introduction to the f77 I/O Library*
fpr(1), fsplit(1), ld(1), ar(1), ranlib(1), dbx(1), intro(3f)
efl(1), ratfor(1), struct(1), prof(1), gprof(1), cc(1)

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

BUGS

Files longer than about 50,000 lines must be split up to be compiled.

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 argument in an attempt to classify it. If an argument appears to be ascii, *file* examines the first 512 bytes and tries to guess its language.

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
find pattern

DESCRIPTION

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

The second form rapidly searches a database for all pathnames which match *pattern*. Usually the database is recomputed weekly and contains the pathnames of all files which are publicly accessible. If escaped, normal shell ''globbing'' characters ('*', '?', '[', and ']') may be used in *pattern*, but the matching differs in that no characters (*e.g.* '/') have to be matched explicitly. As a special case, a simple *pattern* containing no globbing characters is matched as though it were **pattern**; if any globbing character appears there are no implicit globbing characters.

−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, l** or **s** for block special file, character special file, directory, plain file, symbolic link, or socket.

−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).

−nouser
: True if the file belongs to a user *not* in the /etc/passwd database.

−group gname
: True if the file belongs to group *gname* (group name or numeric group ID).

−nogroup
: True if the file belongs to a group *not* in the /etc/group database.

−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; causes the current pathname to be printed.

−ls
: Always true; causes current pathname to be printed together with its associated statistics. These include (respectively) inode number, size in kilobytes (1024 bytes), protection mode,

number of hard links, user, group, size in bytes, and modification time. 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 ''->''. The format is identical to that of ''ls -gilds'' (note however that formatting is done internally, without executing the ls program).

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

−cpio file Write the current file on the argument *file* in *cpio* format.

−xdev Always true; causes find *not* to traverse down into a file system different from the one on which current *argument* pathname resides.

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).

EXAMPLES

To find all accessible files whose pathname contains 'find':

find find

To typeset all variants of manual pages for 'ls':

vtroff -man 'find '*man*/ls.?''

To remove all files named 'a.out' or '*.o' that have not been accessed for a week:

find / \(−name a.out −o −name '*.o' \) −atime +7 −exec rm {} \;

FILES

/etc/passwd
/etc/group
/usr/lib/find/find.codes coded pathnames database

SEE ALSO

sh(1), test(1), fs(5)
Relevant paper in February, 1983 issue of *;login:*.

BUGS

The first form's syntax is painful, and the second form's exact semantics is confusing and can vary from site to site.

More than one '-newer' option does not work properly.

NAME

finger − user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default *finger* lists the login name, full name, terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX user. (Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.)

A longer format also exists and is used by *finger* whenever a list of people's names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* also in the home directory.

Finger may be used to lookup users on a remote machine. The format is to specify the user as ''user@host.'' If the user name is left off, the standard format listing is provided on the remote machine.

Finger options include:

−m Match arguments only on user name.

−l Force long output format.

−p Suppress printing of the *.plan* files

−s Force short output format.

FILES

/etc/utmp	who file
/etc/passwd	for users names, offices, ...
/usr/adm/lastlog	last login times
~/.plan	plans
~/.project	projects

SEE ALSO

chfn(1), w(1), who(1)

AUTHOR

Earl T. Cohen

BUGS

Only the first line of the *.project* file is printed.

The encoding of the gcos field is UCB dependent − it knows that an office '197MC' is '197M Cory Hall', and that '529BE' is '529B Evans Hall'. It also knows that a four digit office phone number should have a ''x2-'' prepended.

There is no way to pass arguments to the remote machine as *finger* uses an internet standard port.

A user information data base is in the works and will radically alter the way the information that *finger* uses is stored. Finger will require extensive modification when this is implemented.

NAME

fmt − simple text formatter

SYNOPSIS

fmt [name ...]

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, within visual mode of the *ex* editor (e.g. *vi*) the command

```
!}fmt
```

will reformat a paragraph, evening the lines.

SEE ALSO

nroff(1), mail(1)

AUTHOR

Kurt Shoens

BUGS

The program 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 will fold 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 coming to *fold.*

SEE ALSO

expand(1)

BUGS

If underlining is present it may be messed up by folding.

NAME

fp − Functional Programming language compiler/interpreter

SYNOPSIS

fp

DESCRIPTION

Fp is an interpreter/compiler that implements the applicative language proposed by John Backus. It is written in **FRANZ LISP**.

In a functional programming language intent is expressed in a mathematical style devoid of assignment statements and variables. Functions compute by value only; there are no side-effects since the result of a computation depends solely on the inputs.

Fp "programs" consist of *functional expressions* − primitive and user-defined *fp* functions combined by *functional forms.* These forms take functional arguments and return functional results. For example, the composition operator *'@'* takes two functional arguments and returns a function which represents their composition.

There exists a single operation in *fp* − *application.* This operation causes the system to evaluate the indicated function using the single argument as input (all functions are monadic).

GETTING STARTED

Fp invokes the system. *Fp* compiles functions into *lisp*(1) source code; *lisp*(1) interprets this code (the user may compile this code using the liszt (1) compiler to gain a factor of 10 in performance). *Control D* exits back to the shell. *Break* terminates any computation in progress and resets any open file units. *)help* provides a short summary of all user commands.

FILES

/usr/ucb/lisp	the FRANZ LISP interpreter
/usr/ucb/liszt	the liszt compiler
/usr/doc/fp	the User's Guide

SEE ALSO

lisp(1), liszt(1).

The Berkeley FP user's manual, available on-line. The language is described in the August 1978 issue of *CACM* (Turing award lecture by John Backus).

BUGS

If a non-terminating function is applied as the result of loading a file, then control is returned to the user immediately, everything after that position in the file is ignored.

FP incorrectly marks the location of a syntax error on large, multi-line function definitions or applications.

AUTHOR

Scott B. Baden

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:

Character	Vertical Space Before Printing
Blank	One line
0	Two lines
1	To first line of next page
+	No advance

A blank 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.

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. If the -s option is given, then only headers for mail sent by *sender* are printed.

FILES

/usr/spool/mail/*

SEE ALSO

biff(1), mail(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.*

AUTHOR

Asa Romberger and Jerry Berkman

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 − ARPANET file transfer program

SYNOPSIS

ftp [**−v**] [**−d**] [**−i**] [**−n**] [**−g**] [**host**]

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer 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* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user the prompt ''ftp>'' is provided to the user. The following commands are recognized by *ftp*:

! [*command* [*args*]]
Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

$ *macro-name* [*args*]
Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

account [*passwd*]
Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any *ntrans* or *nmap* setting. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

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*. An end of file will also terminate the session and exit.

case Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage return stripping during ascii type file retrieval. Records are denoted by a carriage return/linefeed sequence during ascii type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ascii type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.

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, or *local-file* is **-**, output comes to the terminal.

disconnect
A synonym for **close**.

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, subject to alteration by the current *case*, *ntrans*, and *nmap* settings. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

glob Toggle filename expansion for **mdelete**, **mget** and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and not expanded. Globbing for **mput** is done as in **csh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by doing ‘**mls** *remote-files* **-**’. Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a **tar**(1) archive of the subtree (in binary mode).

hash Toggle hash-sign (‘‘#’’) printing for each data block transferred. The size of a data block is 1024 bytes.

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, or if *local-file* is **-**, the output is sent to the terminal.

macdef *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets '$' and '\' as special characters. A '$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '$' followed by an 'i' signals that macro processor that the executing macro is to be looped. On the first pass '$i' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the

second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '$'.

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
Like **dir**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*
Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to *case*, *ntrans*, and *nmap* settings. Files are transferred into the local working directory, which can be changed with '**lcd** directory'; new local directories can be created with '**!** mkdir directory'.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like **ls**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is ''stream'' mode.

mput *local-files*
Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to *ntrans* and *nmap* settings.

nmap [*inpattern outpattern*]
Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences '$1', '$2', ..., '$9' in *inpattern*. Use '\' to prevent this special treatment of the '$' character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values. For exmaple, given *inpattern* $1.$2 and the remote file name "mydata.data", $1 would have the value "mydata", and $2 would have the value "data". The *outpattern* determines the resulting mapped filename. The sequences '$1', '$2', ..., '$9' are replaced by any value resulting from the *inpattern* template. The sequence '$0' is replace by the original filename. Additionally, the sequence '[*seq1*,*seq2*]' is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command "nmap $1.$2.$3 [$1,$2].[$2,file]" would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename ".myfile". Spaces may be included in *outpattern*, as in the example: nmap $1 |sed "s/ *$//" > $1 . Use the '\' character to prevent special treatment of the '$', '[', ']', and ',' characters.

ntrans [*inchars* [*outchars*]]
Set or unset the filename character translation mechanism. If no arguments are specified, the

filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

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 is on), any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

proxy *ftp-command*
Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

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 after processing according to any *ntrans* or *nmap* settings 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.

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*.

reset Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be neccesary following a violation of the ftp protocol by the remote server.

rmdir *directory-name*
Delete a directory on the remote machine.

runique
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
A synonym for put.

sendport
Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default ''stream'' structure is used.

sunique
Toggle storing of files on remote machine under unique file names. Remote ftp server must support ftp protocol STOU command for successful completion. The remote server will report unique name. Default value is off.

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. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. 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 embedded spaces may be quoted with quote (") marks.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name ‘‘–’’ is specified, 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*(3) 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, if ‘‘globbing’’ is enabled, local file names are expanded according to the rules used in the *csh*(1); c.f. the *glob* command. If the *ftp* command expects a single local file (.e.g. **put**), only the first filename generated by the "globbing" operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

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, plus local byte size 8 for **tenex** mode transfers.

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.

The **–v** (verbose on) option forces *ftp* to show all responses from the remote server, as well as report on data transfer statistics.

The **–n** option restrains *ftp* from attempting ‘‘auto-login’’ upon initial connection. If auto-login is enabled, *ftp* will check the *.netrc* (see below) file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

The **–i** option turns off interactive prompting during multiple file transfers.

The **–d** option enables debugging.

The **–g** option disables file name globbing.

THE .netrc FILE

The .netrc file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

machine *name*

Identify a remote machine name. The auto-login process searches the .netrc file for a **machine** token that matches the remote machine specified on the *ftp* command line or as an **open** command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the end of file is reached or another **machine** token is encountered.

login *name*
Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the .netrc file, *ftp* will abort the auto-login process if the .netrc is readable by anyone besides the user.

account *string*
Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

macdef *name*
Define a macro. This token functions like the *ftp* **macdef** command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a null line (consecutive new-line characters) is encountered. If a macro named *init* is defined, it is automatically executed as the last step in the auto-login process.

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD UNIX ascii-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the ascii type. Avoid this problem by using the binary image type.

NAME

gcore − get core images of running processes

SYNOPSIS

gcore process-id ...

DESCRIPTION

Gcore creates a core image of each specified process, suitable for use 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

gprof − display call graph profile data

SYNOPSIS

gprof [options] [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 which are 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.

The following options are available:

−a suppresses the printing of statically declared functions. If this option is given, all relevant information about the static function (*e.g.,* 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 supresses the printing of 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)
''gprof: A Call Graph Execution Profiler'', by Graham, S.L., Kessler, P.B., McKusick, M.K.; *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 [option] ...

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.

The following options are recognized, each as a separate argument.

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument 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 Character string given by next argument is default label for each point.

−g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

−l Next argument is label for graph.

−m Next argument 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**] If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

−y [**l**] Similarly for *y*.

−h Next argument is fraction of space for height.

−w Similarly for width.

−r Next argument is fraction of space to move right before plotting.

−u Similarly to move up before plotting.

−t Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present.

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 [option] ... expression [file] ...

egrep [option] ... [expression] [file] ...

fgrep [option] ... [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); it uses a compact nondeterministic algorithm. *Egrep* patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed strings; it is fast and compact. The following options are recognized.

−v All lines but those matching are printed.

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

−c Only a count of matching lines is printed.

−l The names of files with matching lines are listed (once) separated by newlines.

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

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

−i The case of letters is ignored in making comparisons — that is, upper and lower case are considered identical. This applies to *grep* and *fgrep* only.

−s Silent mode. Nothing is printed (except error messages). This is useful for checking the error status.

−w The expression is searched for 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* The regular expression (*egrep*) or string list (*fgrep*) is taken from the *file.*

In all cases the file name is shown if there is more than one input file. Care should be taken when using the characters $ * [^ | ( ) and \ in the *expression* as they are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes ´ ´.

Fgrep searches for lines that contain one of the (newline-separated) *strings.*

Egrep accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character other than newline matches that character.

The character ^ matches the beginning of a line.

The character $ matches the end of a line.

A **.** (period) matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the 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 regular expression followed by an * (asterisk) matches a sequence of 0 or more matches of the regular expression. A regular expression followed by a + (plus) matches a sequence of 1 or more matches of the regular expression. A regular expression followed by a ? (question mark) matches a sequence of 0 or 1 matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

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

The order of precedence of operators at the same parenthesis level is [ ] then *+? then concatenation then | and newline.

Ideally there should be only one *grep,* but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

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.

NAME

groups − show group memberships

SYNOPSIS

groups [user]

DESCRIPTION

The *groups* command shows the groups to which you or the optionally specified user 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

BUGS

More groups should be allowed.

NAME

head − give first few lines

SYNOPSIS

head [**−**count] [file ...]

DESCRIPTION

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

SEE ALSO

tail(1)

NAME

hostid − set or print identifier of current host system

SYNOPSIS

hostid [identifier]

DESCRIPTION

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

SEE ALSO

gethostid(2), sethostid(2)

NAME

hostname − set or print name of current host system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

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

SEE ALSO

gethostname(2), sethostname(2)

NAME

indent − indent and format C program source

SYNOPSIS

indent [*input-file* [*output-file*]] [**−bad** | **−nbad**] [**−bap** | **−nbap**] [**−bbb** | **−nbbb**] [**−bc** | **−nbc**] [**−bl** | **−br**] [**−c***n*] [**−cd***n*] [**−cdb** | **−ncdb**] [**−ce** | **−nce**] [**−ci***n*] [**−cli***n*] [**−d***n*] [**−di***n*] [**−dj** | **−ndj**] [**−ei** | **−nei**] [**−fc1** | **−nfc1**] [**−i***n*] [**−ip** | **−nip**] [**−l***n*] [**−lc***n*] [**−lp** | **−nlp**] [**−npro**] [**−pcs** | **−npcs**] [**−ps** | **−nps**] [**−psl** | **−npsl**] [**−sc** | **−nsc**] [**−sob** | **−nsob**] [**−st**] [**−troff**] [**−v** | **−nv**]

DESCRIPTION

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

NOTE: If you only specify an *input-file*, the formatting is done 'in-place', that is, the formatted file is written back into *input-file* and a backup copy of *input-file* is written in the current directory. If *input-file* is named '/blah/blah/file', the backup file is named file*.BAK.*

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

OPTIONS

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

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

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

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

−bc,**−nbc** If **−bc** is specified, then a newline is forced after each comma in a declaration. **−nbc** turns off this option. The default is **−nbc**.

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

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

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

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

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

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

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

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

Rather than like this:

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

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

−ce, **−nce** Enables (disables) forcing 'else's to cuddle up to the immediately preceding '}'. The default is **−ce**.

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

−cli*n* Causes case labels to be indented *n* tab stops to the right of the containing **switch** statement. **−cli0.5** causes case labels to be indented half a tab stop. The default is **−cli0**. (This is the only option that takes a fractional argument.)

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

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

−dj,**−ndj** **−dj** left justifies declarations. **−ndj** indents declarations the same as code. The default is **−ndj**.

−ei, **−nei** Enables (disables) special **else-if** processing. If enabled, **if**s following **else**s will have the same indentation as the preceding **if** statement. The default is **−ei**.

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

−i*n* The number of spaces for one indentation level. The default is 8.

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

−l*n* Maximum length of an output line. The default is 78.

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

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

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

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

Inserting two more newlines we get:

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

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

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

−ps,**−nps** If true (**−ps**) the pointer following operator '−>' will be surrounded by spaces on either side. The default is **−nps**.

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

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

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

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

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

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

−v,**−nv** **−v** turns on 'verbose' mode; **−nv** turns it off. When in verbose mode, *indent* reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is **−nv**.

FURTHER DESCRIPTION

You may set up your own 'profile' of defaults to *indent* by creating a file called ***.indent.pro*** in either your login directory and/or the current directory and including whatever switches you like. Switches in '.indent.pro' in the current directory override those in your login directory (with the exception of **-T** type definitions, which just accumulate). If *indent* is run and a profile file exists, then it is read to set up the program's defaults. The switches should be separated by spaces, tabs or newlines. Switches on the command line, however, override profile switches.

Comments

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

Straight text. All other comments are treated as straight text. *Indent* fits as many words (separated by blanks, tabs, or newlines) on a line as possible. Blank lines break paragraphs.

Comment indentation

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

Preprocessor lines

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

C syntax

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

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

is handled properly.

FILES

./.indent.pro	profile file
~/.indent.pro	profile file

BUGS

Indent has even more switches than *ls*.

A common mistake that often causes grief is typing:

```
indent *.c
```

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

NAME

install − install binaries

SYNOPSIS

install [**−c**] [**−m** mode] [**−o** owner] [**−g** group] [**−s**] binary destination

DESCRIPTION

Binary is moved (or copied if **−c** is specified) to *destination*. If *destination* already exists, it is removed before *binary* is moved. If the destination is a directory then *binary* is moved into the *destination* directory with its original file-name.

The mode for *Destination* is set to 755; the **−m** *mode* option may be used to specify a different mode.

Destination is changed to owner root; the **−o** *owner* option may be used to specify a different owner.

Destination is changed to group staff; the **−g** *group* option may be used to specify a different group.

If the **−s** option is specified the binary is stripped after being installed.

Install refuses to move a file onto itself.

SEE ALSO

chgrp(1), chmod(1), cp(1), mv(1), strip(1), chown(8)

NAME

iostat − report I/O statistics

SYNOPSIS

iostat [drives] [interval [count]]

DESCRIPTION

Iostat iteratively reports the number of characters read and written to terminals per second, and, for each disk, the number of transfers per second, kilobytes transferred per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices it is possible to determine average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

If more than 4 disk drives are configured in the system, *iostat* displays only the first 4 drives, with priority given to Massbus disk drives (i.e. if both Unibus and Massbus drives are present and the total number of drives exceeds 4, then some number of Unibus drives will not be displayed in favor of the Massbus drives). To force *iostat* to display specific drives, their names may be supplied on the command line.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(1)

NAME

join − relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

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

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

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

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

These options are recognized:

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

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

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

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

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

SEE ALSO

sort(1), comm(1), awk(1)

BUGS

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

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

NAME

jove − an interactive display-oriented text editor

SYNOPSIS

jove [-d directory] [-w] [-t tag] [+n file] [-p file] [files]
jove -r

DESCRIPTION

JOVE is Jonathan's Own Version of Emacs. It is based on the original EMACS editor written at MIT by Richard Stallman. Although JOVE is meant to be compatible with EMACS, there are some major differences between the two editors and you shouldn't rely on their behaving identically.

JOVE works on any reasonable display terminal that is described in the *termcap* file (see TERMCAP(5) for more details). When you start up JOVE, it checks to see whether you have your *TERM* environment variable set. On most systems that will automatically be set up for you, but if it's not JOVE will ask you what kind of terminal you are using. To avoid having to type this every time you run JOVE you can set your *TERM* environment variable yourself. How you do this depends on which shell you are running. If you are running the C Shell, as most of you are, you type

% setenv TERM *type*

and with the Bourne Shell, you type

$ TERM= *type* ; export TERM

where *type* is the name of the kind of terminal you are using (e.g., vt100). If neither of these works get somebody to help you.

INVOKING JOVE

If you run JOVE with no arguments you will be placed in an empty buffer, called *Main.* Otherwise, any arguments you supply are considered file names and each is "given" its own buffer. Only the first file is actually read in--reading other files is deferred until you actually try to use the buffers they are attached to. This is for efficiency's sake: most of the time, when you run JOVE on a big list of files, you end up editing only a few of them.

The names of all of the files specified on the command line are saved in a buffer, called *Minibuf.* The mini-buffer is a special JOVE buffer that is used when JOVE is prompting for some input to many commands (for example, when JOVE is prompting for a file name). When you are being prompted for a file name, you can type C-N (that's Control-N) and C-P to cycle through the list of files that were specified on the command line. The file name will be inserted where you are typing and then you can edit it as if you typed it in yourself.

JOVE recognizes the following switches:

-d The following argument is taken to be the name of the current directory. This is for systems that don't have a version of C shell that automatically maintains the *CWD* environment variable. If *-d* is not specified on a system without a modified C shell, JOVE will have to figure out the current directory itself, and that can be VERY slow.

+n Reads the file, designated by the following argument, and positions point at the *n'th* line instead of the (default) 1'st line. This can be specified more than once but it doesn't make sense to use it twice on the same file; in that case the second one wins.

-p Parses the error messages in the file designated by the following argument. The error messages are assumed to be in a format similar to the C compiler, LINT, or GREP output.

-t Runs the *find-tag* command on the following argument (see ctags(1)).

-w Divides the window in two. When this happens, either the same file is displayed in both windows, or the second file in the list is read in and displayed in its window.

As a special case, invoking JOVE with the -r option runs JOVE_RECOVER. Use this when the system crashes, or JOVE crashes, or you accidently get logged out while in JOVE. If there are any buffers to be recovered, this will find them. Read the documentation for JOVE_RECOVER.

GETTING HELP

Once in JOVE, there are several commands available to get help. To execute any JOVE command, you type "<ESC> X command-name" followed by <Return>. To get a list of all the JOVE commands you type "<ESC> X" followed by "?". The *describe-bindings* command can be used to get a list containing each key, and its associated command (that is, the command that gets executed when you type that key). If you want to save the list of bindings, you can set the jove variable *send-typeout-to-buffer* to ON (using the *set* command), and then execute the *describe-bindings* command. This will create a buffer and put in it the bindings list it normally would have printed on the screen. Then you can save that buffer to a file and print it to use as a quick reference card. (See VARIABLES below.)

Once you know the name of a command, you can find out what it does with the *describe-command* command, which you can invoke quickly by typing "ESC ?". The *apropos* command will give you a list of all the command with a specific string in their names. For example, if you want to know the names of all the commands that are concerned with windows, you can run "apropos" with the keyword *window.*

If you're not familar with the EMACS command set, it would be worth your while to use run TEACHJOVE. Do do that, just type "teachjove" to your shell and you will be placed in JOVE in a file which contains directions. I highly recommend this for beginners; you may save yourself a lot of time and headaches.

KEY BINDINGS and VARIABLES

You can alter the key bindings in JOVE to fit your personal tastes. That is, you can change what a key does every time you strike it. For example, by default the C-N key is bound to the command *next-line* and so when you type it you move down a line. If you want to change a binding or add a new one, you use the *bind-to-key* command. The syntax is "bind-to-key <command> key".

You can also change the way JOVE behaves in little ways by changing the value of some variables with the *set* command. The syntax is "set <variable> value", where value is a number or a string, or "on" or "off", depending on the context. For example, if you want JOVE to make backup files, you set the "make-backup-files" variable to "on". To see the value of a variable, use the "print <variable>" command.

INITIALIZATION

JOVE automatically reads commands from an initialization file in your HOME directory, called ".joverc". In this file you can place commands that you would normally type in JOVE. If you like to rearrange the key bindings and set some variables every time you get into JOVE, you should put them in your initialization file. Here are a few lines from mine:

```
set match-regular-expressions on
auto-execute-command auto-fill /tmp/Re\|.*drft
bind-to-key i-search-forward ^\
bind-to-key i-search-reverse ^R
bind-to-key find-tag-at-point ^[^T
bind-to-key scroll-down ^C
bind-to-key grow-window ^Xg
bind-to-key shrink-window ^Xs
```

(Note that the Control Characters can be either two character sequences (e.g. ^ and C together as ^C) or the actual control character. If you want to use an ^ by itself you must BackSlash it (e.g., bind-to-key grow-window ^X\^ binds grow-window to "^X^").

SOME MINOR DETAILS

You should type C-\ instead of C-S in many instances. For example, the way to search for a string is documented as being "C-S" but in reality you should type "C-\". This is because C-S is the XOFF character (what gets sent when you type the NO SCROLL key), and clearly that won't work. The XON character is "C-Q" (what gets sent when you type NO SCROLL again) which is documented as the way to do a quoted-insert. The alternate key for this is "C-^" (typed as "C-`" on vt100's and its look-alikes). If you want to enable C-S and C-Q and you know what you are doing, you can put the line:

    set allow-^S-and-^Q on

in your initialization file.

If your terminal has a metakey, JOVE will use it if you turn on the "meta-key" variable. JOVE will automatically turn on "meta-key" if the METAKEY environment variable exists. This is useful for if you have different terminals (e.g., one at home and one at work) and one has a metakey and the other doesn't.

FILES

/usr/new/lib/jove/.joverc - system wide initialization file
~/.joverc - personal initialization file
/tmp - where temporary files are stored
/usr/new/lib/jove/teach-jove - the interactive tutorial
/usr/new/lib/jove/portsrv - for running shells in windows (pdp11 only)

SEE ALSO

jove_recover(1N) - to recover buffers after a system/editor crash
ed(1) - for a description of regular expressions
teachjove(1N) - for an interactive JOVE tutorial.
JOVE Manual for UNIX Users
(USD:17)

DIAGNOSTICS

JOVE diagnostics are meant to be self-explanatory, but you are advised to seek help whenever you are confused. You can easily lose a lot of work if you don't know EXACTLY what you are doing.

BUGS

Lines can't be more than 1024 characters long.

Searches can't cross line boundaries.

AUTHOR

Jonathan Payne

NAME

kill − terminate a process with extreme prejudice

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 'kill −l', 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; '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 (i.e. processes resulting from the current login) are signaled (but beware: this works only if you use *sh*(1); not if you use *csh*(1).) Negative process numbers also have special meanings; see *kill*(2) for details.

The killed processes must belong to the current user unless he is the super-user.

The process number of an asynchronous process started with '&' is reported by the shell. Process numbers can also be found by using *ps*(1). *Kill* is a built-in to *csh*(1); it allows job specifiers of the form ''%...'' as arguments so process id's are not as often used as *kill* arguments. See *csh*(1) for details.

SEE ALSO

csh(1), ps(1), kill(2), sigvec(2)

BUGS

A replacement for ''kill 0'' for *csh*(1) users should be provided.

NAME

last − indicate last logins of users and teletypes

SYNOPSIS

last [−N] [name ...] [tty ...]

DESCRIPTION

Last will look 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* will print 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 prints a record of all logins and logouts, in reverse order. The **−N** option limits the report to N lines.

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.

FILES

/usr/adm/wtmp	login data base
/usr/adm/shutdownlog	which records shutdowns and reasons for same

SEE ALSO

wtmp(5), ac(8), lastcomm(1)

AUTHOR

Howard Katseff

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. With no arguments, *lastcomm* prints information about all the commands recorded during the current accounting file's lifetime. If called with arguments, only accounting entries with a matching command name, user name, or terminal name are printed. So, for example,

lastcomm a.out root ttyd0

would produce a listing of all the executions of commands named *a.out* by user *root* on the terminal *ttyd0*.

For each process entry, the following are printed.

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.

The flags are encoded as follows: ''S'' indicates the command was executed by the super-user, ''F'' indicates the command ran after a fork, but without a following *exec*, ''C'' indicates the command was run in PDP-11 compatibility mode (VAX only), ''D'' indicates the command terminated with the generation of a *core* file, and ''X'' indicates the command was terminated with a signal.

FILES

/usr/adm/acct

SEE ALSO

last(1), sigvec(2), acct(8), core(5)

NAME

ld − link editor

SYNOPSIS

ld [option] ... 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.out**. This file is made executable only if no errors occurred during the load.

The argument routines 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 argument 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.

Ld understands several options. Except for **−l**, they should appear before the file names.

−A This option specifies incremental loading, i.e. 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 will be taken as a basis on which to define additional symbols. Only newly linked material will be 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 1024). The default value is the old value of _end.

−D Take the next argument as a hexadecimal number and pad the data segment with zero bytes to the indicated length.

−d Force definition of common storage even if the **−r** flag is present.

−e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

−L*dir* Add *dir* to the list of directories in which libraries are searched for. Directories specified with **−L** are searched before the standard directories.

−l*x* This option is an abbreviation for the library name 'lib*x*.a', where *x* is a string. *Ld* searches for libraries first in any directories specified with **−L** options, then in the standard directories '/lib', '/usr/lib', and '/usr/local/lib'. 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 1024 byte boundary following the end of the text.

−o The *name* argument after **−o** is used as 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.

−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 The next argument is a hexadecimal number which sets the text segment origin. The default origin is 0.

−t ("trace") Print the name of each file as it is processed.

−u Take the following argument as a symbol and enter it as undefined in the symbol table. 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* Indicate each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. (It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.)

−z Arrange for the process to be loaded on demand from the resulting executable file (413 format) rather than preloaded. This is the default. Results in a 1024 byte header on the output file followed by a text and data segment each of which have size a multiple of 1024 bytes (being padded out with nulls in the file if necessary). With this format the first few BSS segment symbols may actually appear (from the output of *size*(1)) to live in the data segment; this 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)

BUGS

There is no way to force data to be page aligned. *Ld* pads images which are to be demand loaded from the file system to the next page boundary to avoid a bug in the system.

NAME

learn − computer aided instruction about UNIX

SYNOPSIS

learn [−directory] [subject [lesson]]

DESCRIPTION

Learn gives Computer Aided Instruction courses and practice in the use of UNIX, the C Shell, and the Berkeley text editors. To get started simply type **learn**. If you had used *learn* before and left your last session without completing a subject, the program will use information in $HOME/.learnrc to start you up in the same place you left off. Your first time through, *learn* will ask questions to find out what you want to do. Some questions may be bypassed by naming a *subject*, and more yet by naming a *lesson*. You may enter the *lesson* as a number that *learn* gave you in a previous session. If you do not know the lesson number, you may enter the *lesson* as a word, and *learn* will look for the first lesson containing it. If the *lesson* is '−', *learn* prompts for each lesson; this is useful for debugging.

The *subject* 's presently handled are

files
editor
vi
morefiles
macros
eqn
C

There are a few special commands. The command 'bye' terminates a *learn* session and 'where' tells you of your progress, with 'where m' telling you more. The command 'again' re-displays the text of the lesson and 'again *lesson*' lets you review *lesson*. There is no way for *learn* to tell you the answers it expects in English, however, the command 'hint' prints the last part of the lesson script used to evaluate a response, while 'hint m' prints the whole lesson script. This is useful for debugging lessons and might possibly give you an idea about what it expects.

The −*directory* option allows one to exercise a script in a nonstandard place.

FILES

/usr/lib/learn	subtree for all dependent directories and files
/usr/tmp/pl*	playpen directories
$HOME/.learnrc	startup information

SEE ALSO

csh(1), ex(1)

B. W. Kernighan and M. E. Lesk, *LEARN − Computer-Aided Instruction on UNIX*

BUGS

The main strength of *learn,* that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Occasionally a lesson script does not recognize all the different correct responses, in which case the 'hint' command may be useful. Such lessons may be skipped with the 'skip' command, but it takes some sophistication to recognize the situation.

To find a *lesson* given as a word, *learn* does a simple *fgrep*(1) through the lessons. It is unclear whether this sort of subject indexing is better than none.

Spawning a new shell is required for each of many user and internal functions.

The 'vi' lessons are provided separately from the others. To use them see your system administrator.

NAME

leave − remind you when you have to leave

SYNOPSIS

leave [[+]hhmm]

DESCRIPTION

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. When you log off, *leave* exits just before it would have printed the next message.

The time of day is in the form hhmm where hh is a time in hours (on a 12 or 24 hour clock). All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If the time is preceeded by '+', the alarm will go off in hours and minutes from the current time.

If no argument is given, *leave* prompts with "When do you have to leave?". A reply of newline causes *leave* to exit, 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'' giving its process id.

SEE ALSO

calendar(1)

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.

The options have the following meanings.

−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 a *lex* program that would be put into a *lex* command file. 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)
M. E. Lesk and E. Schmidt, *LEX − Lexical Analyzer Generator*

NAME

lint − a C program verifier

SYNOPSIS

lint [**−abchnpuvx**] 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). Arguments ending in *.ln* are also treated as library files. To create lint libraries, use the **−C** option:

lint −Cfoo files . . .

where *files* are the C sources of library *foo.* The result is a file *llib-lfoo.ln* in the correct library format suitable for linting programs using *foo.*

Any number of the options in the following list may be used. The **−D**, **−U**, and **−I** options of *cc*(1) are also recognized as separate arguments.

p Attempt to check portability to the *IBM* and *GCOS* dialects of C.

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.

z Do not complain about structures that are never defined (e.g. using a structure pointer without knowing its contents.).

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.

AUTHOR

S.C. Johnson. Lint library construction implemented by Edward Wang.

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-port.ln	declarations for portable functions
/usr/lib/lint/llib-port	human readable . . .
llib-l*.ln	library created with **−C**

SEE ALSO

cc(1)
S. C. Johnson, *Lint, a C Program Checker*

BUGS

There are some things you just **can't** get lint to shut up about.

/*NOSTRICT*/ is not implemented in the current version (alas).

NAME

lisp − lisp interpreter

SYNOPSIS

lisp

DESCRIPTION

Lisp is a lisp interpreter for a dialect which closely resembles MIT's MACLISP. This lisp, known as FRANZ LISP, features an I/O facility which allows the user to change the input and output syntax, add macro characters, and maintain compatibility with upper-case only lisp systems; infinite precision integer arithmetic, and an error facility which allows the user to trap system errors in many different ways. Interpreted functions may be mixed with code compiled by *liszt*(1) and both may be debugged using the ''Joseph Lister'' trace package. A *lisp* containing compiled and interpreted code may be dumped into a file for later use.

There are too many functions to list here; one should refer to the manuals listed below.

AUTHORS

An early version was written by Jeff Levinsky, Mike Curry, and John Breedlove. Keith Sklower wrote and is maintaining the current version, with the assistance of John Foderaro. The garbage collector was implemented by Bill Rowan.

FILES

/usr/lib/lisp/trace.l	Joseph Lister trace package
/usr/lib/lisp/toplevel.l	top level read-eval-print loop

SEE ALSO

liszt(1), lxref(1)
'FRANZ LISP Manual, Version 1' by John K. Foderaro
MACLISP Manual

BUGS

The error system is in a state of flux and not all error messages are as informative as they could be.

NAME

liszt − compile a Franz Lisp program

SYNOPSIS

liszt [**−mpqruwxCQST**] [**−e** form] [**−o** objfile] [name]

DESCRIPTION

Liszt takes a file whose names ends in '.l' and compiles the FRANZ LISP code there leaving an object program on the file whose name is that of the source with '.o' substituted for '.l'.

The following options are interpreted by *liszt.*

−e Evaluate the given form before compilation begins.

−m Compile a MACLISP file, by changing the readtable to conform to MACLISP syntax and including a macro-defined compatibility package.

−o Put the object code in the specified file, rather than the default '.o' file.

−p places profiling code at the beginning of each non-local function. If the lisp system is also created with profiling in it, this allows function calling frequency to be determined (see *prof*(1).)

−q Only print warning and error messages. Compilation statistics and notes on correct but unusual constructs will not be printed.

−r place bootstrap code at the beginning of the object file, which when the object file is executed will cause a lisp system to be invoked and the object file fasl'ed in.

−u Compile a UCI-lispfile, by changing the readtable to conform to UCI-Lisp syntax and including a macro-defined compatibility package.

−w Suppress warning diagnostics.

−x Create a lisp cross reference file with the same name as the source file but with '.x' appended. The program *lxref*(1) reads this file and creates a human readable cross reference listing.

−C put comments in the assembler output of the compiler. Useful for debugging the compiler.

-Q Print compilation statistics and warn of strange constructs. This is the default.

−S Compile the named program and leave the assembler-language output on the corresponding file suffixed '.s'. This will also prevent the assembler language file from being assembled.

−T send the assembler output to standard output.

If no source file is specified, then the compiler will run interactively. You will find yourself talking to the *lisp*(1) top-level command interpreter. You can compile a file by using the function *liszt* (an nlambda) with the same arguments as you use on the command line. For example to compile 'foo', a MACLISP file, you would use:

(liszt −m foo)

Note that *liszt* supplies the ''.l'' extension for you.

FILES

/usr/lib/lisp/machacks.l	MACLISP compatibility package
/usr/lib/lisp/syscall.l	macro definitions of Unix system calls
/usr/lib/lisp/ucifnc.l	UCI Lisp compatibility package

AUTHOR

John Foderaro

SEE ALSO

lisp(1), lxref(1)

NAME

ln − make links

SYNOPSIS

ln [**−s**] sourcename [targetname]
ln [**−s**] sourcename1 sourcename2 [sourcename3 ...] targetdirectory

DESCRIPTION

A link is a directory entry referring to a file; the same file (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.

By default *ln* makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

The **−s** option causes *ln* to create symbolic links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open*(2) operation is performed on the link. A *stat*(2) on a symbolic link will return 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.

Given one or two arguments, *ln* creates a link to an existing file *sourcename*. If *targetname* is given, the link has that name; *targetname* may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of *sourcename*.

Given more than two arguments, *ln* makes links in *targetdirectory* to all the named source files. The links made will have the same name as the files being linked to.

SEE ALSO

rm(1), cp(1), mv(1), link(2), readlink(2), stat(2), symlink(2)

NAME

lock − reserve a terminal

SYNOPSIS

lock [**−number**]

DESCRIPTION

Lock requests a password from the user, reads it again for verification and then it will normally not relinquish the terminal until the password is repeated. There are three other conditions under it will terminate: it accepts the password for root as an alternative to the one given by the user, it will timeout after some interval of time, and it may be killed by somebody with the appropiate permission. The default time limit is 15 minutes but it may be changed with the **−number** option where *number* is the time limit in minutes.

NAME

logger − make entries in the system log

SYNOPSIS

logger [**−t** tag] [**−p** pri] [**−i**] [**−f** file] [message ...]

ARGUMENTS

−t *tag* Mark every line in the log with the specified *tag*.

−p *pri* Enter the message with the specified priority. The priority may be specified numerically or as a ''facility.level'' pair. For example, ''−p local3.info'' logs the message(s) as *info*rmational level in the *local3* facility. The default is ''user.notice.''

−i Log the process id of the logger process with each line.

−f *file* Log the specified file.

message The message to log; if not specified, the **−f** file or standard input is logged.

DESCRIPTION

Logger provides a program interface to the *syslog*(3) system log module.

A message can be given on the command line, which is logged immediately, or a file is read and each line is logged.

EXAMPLES

logger System rebooted

logger −p local0.notice −t HOSTIDM −f /dev/idmc

SEE ALSO

syslog(3), syslogd(8)

NAME

login − sign on

SYNOPSIS

login [−p] [username]

DESCRIPTION

The *login* command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See ''How to Get Started'' for how to dial up initially.

If *login* is invoked without an argument, it asks for a user name, and, if appropriate, a password. 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 and the user is informed of the existence of mail. The message of the day is printed, as is the time of his last login. Both are suppressed if he has a ''.hushlogin'' file in his home directory; this is mostly 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 *csh*(1)) according to specifications found in a password file. Argument 0 of the command interpreter is the name of the command interpreter with a leading dash (''−'').

Login also modifies the environment *environ*(7) with information specifying home directory, command interpreter, terminal type (if available) and user name. The '−p' argument causes the remainder of the environment to be preserved, otherwise any previous environment is discarded.

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 users logging in when the system is about to go down.

Login is recognized by *sh*(1) and *csh*(1) and executed directly (without forking).

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/spool/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/nologin	stops logins
.hushlogin	makes login quieter

SEE ALSO

init(8), getty(8), mail(1), passwd(1), passwd(5), environ(7), shutdown(8), rlogin(1c)

DIAGNOSTICS

''Login incorrect,'' if the name or the password is bad.
''No Shell'', ''cannot open password file'', ''no directory'': consult a programming counselor.

BUGS

An undocumented option, **−r** is used by the remote login server, *rlogind*(8C) to force *login* to enter into an initial connection protocol. **−h is used by** *telnetd*(8C) and other servers to list the host from which the connection was received.

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*. It uses binary search.

The options **d** and **f** affect comparisons as in *sort*(1):

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, */usr/dict/words* is assumed with collating sequence **−df.**

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

indxbib, lookbib − build inverted index for a bibliography, find references in a bibliography

SYNOPSIS

indxbib database ...
lookbib [**−n**] 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.

Lookbib will ask if you need instructions, and will print some brief information if you reply ''y''. The ''−n'' flag turns off the prompt for instructions.

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

lorder − find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see *ar*(1)) *files.* The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

This brash one-liner intends to build a new library from existing '.o' files.

```
ar cr library ` lorder *.o | tsort`
```

The need for lorder may be vitiated by use of *ranlib*(1), which converts an ordered archive into a randomly accessed library.

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

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'; nonsense results otherwise.

NAME

lpq − spool queue examination program

SYNOPSIS

lpq [+[n]] [-l] [−Pprinter] [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* invoked without any arguments reports on any jobs currently in the queue. A **−P** flag may be used to specify a particular printer, otherwise the default line printer is used (or the value of the PRINTER variable in the environment). If a + argument is supplied, *lpq* displays the spool queue until it empties. Supplying a number immediately after the + sign indicates that *lpq* should sleep *n* seconds in between scans of the queue. All other arguments supplied are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (i.e. 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 (i.e. due to some malfunction), the *lpc*(8) command can be used to restart the printer daemon.

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

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 results in widely spaced columns.

DIAGNOSTICS

Unable to open various files. The lock file being malformed. Garbage files when there is no daemon active, but files in the spooling directory.

NAME

lpr − off line print

SYNOPSIS

lpr [**−P***printer*] [**−#***num*] [**−C** *class*] [**−J** *job*] [**−T** *title*] [**−i** [*numcols*]] [**−1234** *font*] [**−w***num*] [**−pltndgvcfrmhs**] [name ...]

DESCRIPTION

Lpr uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed. The **−P** option may be used to force output to a specific printer. Normally, the default printer is used (site dependent), or the value of the environment variable PRINTER is used.

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 Use a filter which allows control characters to be printed and suppresses page breaks.

−t The files are assumed to contain data from *troff*(1) (cat phototypesetter commands).

−n The files are assumed to contain data from *ditroff* (device independent troff).

−d The files are assumed to contain data from *tex*(l) (DVI format from Stanford).

−g The files are assumed to 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 are assumed to contain a raster image for devices like the Benson Varian.

−c The files are assumed to contain data produced by *cifplot*(l).

−f Use a filter which interprets the first character of each line as a standard FORTRAN carriage control character.

The remaining single letter options have the following meaning.

−r Remove the file upon completion of spooling or upon completion of printing (with the **−s** option).

−m Send mail upon completion.

−h Suppress the printing of the burst page.

−s Use symbolic links. Usually files are copied to the spool directory.

The **−C** option takes the following argument as a job classification for use on the burst page. For example,

```
lpr −C EECS foo.c
```

causes the system name (the name returned by *hostname*(1)) to be replaced on the burst page by EECS, and the file foo.c to be printed.

The **−J** option takes the following argument as the job name to print on the burst page. Normally, the first file's name is used.

The **−T** option uses the next argument as the title used by *pr*(1) instead of the file name.

To get multiple copies of output, use the **−#***num* option, where *num* is the number of copies desired of each file named. For example,

```
lpr −#3 foo.c bar.c more.c
```

would result in 3 copies of the file foo.c, followed by 3 copies of the file bar.c, etc. On the other hand,

```
cat foo.c bar.c more.c | lpr −#3
```

will give three copies of the concatenation of the files.

The **–i** option causes the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.

The **–w** option takes the immediately following number to be the page width for *pr*.

The **–s** option will 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.

The option **–1234** Specifies a font to be mounted on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size*.

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

If you try to spool too large a file, it will be truncated. *Lpr* will object to printing binary files. If a user other than root prints a file and spooling is disabled, *lpr* will print a message saying so and will not put jobs in the queue. If a connection to *lpd* on the local machine cannot be made, *lpr* will say that the daemon cannot be started. Diagnostics may be printed in the daemon's log file regarding missing spool files by *lpd*.

BUGS

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

NAME

lprm − remove jobs from the line printer spooling queue

SYNOPSIS

lprm [**−P***printer*] [**−**] [job # ...] [user ...]

DESCRIPTION

Lprm will remove 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, e.g.

```
% lpq –l

1st: ken                    [job #013ucbarpa]
     (standard input)       100 bytes
% lprm 13
```

Lprm will announce 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 usd 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

''Permission denied" if the user tries to remove files other than his own.

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

lptest − generate lineprinter ripple pattern

SYNOPSIS

lptest [**length** [**count**]]

DESCRIPTION

Lptest writes the traditional "ripple test" pattern on standard output. In 96 lines, this pattern will print all 96 printable ASCII characters in each position. While originally created to test printers, it is quite useful for testing terminals, driving terminal ports for debugging purposes, or any other task where a quick supply of random data is needed.

The *length* argument specifies the output line length if the the default length of 79 is inappropriate.

The *count* argument specifies the number of output lines to be generated if the default count of 200 is inappropriate. Note that if *count* is to be specified, *length* must be also be specified.

SEE ALSO

BUGS

NAME

ls − list contents of directory

SYNOPSIS

ls [**−acdfgilqrstu1ACLFR**] name ...

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. 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.

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 cause directories to be marked with a trailing ‘/’, sockets 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.

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 a 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 refers 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

lxref − lisp cross reference program

SYNOPSIS

lxref [**−N**] xref-file ... [**−a** source-file ...]

DESCRIPTION

Lxref reads cross reference file(s) written by the lisp compiler *liszt* and prints a cross reference listing on the standard output. *Liszt* will create a cross reference file during compilation when it is given the **−x** switch. Cross reference files usually end in '.x' and consequently *lxref* will append a '.x' to the file names given if necessary. The first option to *lxref* is a decimal integer, N, which sets the *ignorelevel.* If a function is called more than *ignorelevel* times, the cross reference listing will just print the number of calls instead of listing each one of them. The default for *ignorelevel* is 50.

The **−a** option causes *lxref* to put limited cross reference information in the sources named. *lxref* will scan the source and when it comes across a definition of a function (that is a line beginning with *'(def'* it will preceed that line with a list of the functions which call this function, written as a comment preceeded by *';.. '* . All existing lines beginning with *';.. '* will be removed from the file. If the source file contains a line beginning *';.-'* then this will disable this annotation process from this point on until a *';.+'* is seen (however, lines beginning with *';.. '* will continue to be deleted). After the annoation is done, the original file *'foo.l'* is renamed to " *'#.foo.l'* " and the new file with annotation is named *'foo.l'*

AUTHOR

John Foderaro

SEE ALSO

lisp(1), liszt(1)

BUGS

NAME

m4 − macro processor

SYNOPSIS

m4 [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no arguments, or if an argument is '−', the standard input is read. 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 (i.e., \` ´).

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

B. W. Kernighan and D. M. Ritchie, *The M4 Macro Processor*

NAME

mail − send and receive mail

SYNOPSIS

mail [**−v**] [**−i**] [**−n**] [**−s** subject] [user ...]
mail [**−v**] [**−i**] [**−n**] **−f** [name]
mail [**−v**] [**−i**] [**−n**] **−u** user

INTRODUCTION

Mail is a intelligent mail processing system, which has a command syntax reminiscent of *ed* with lines replaced by messages.

The **−v** flag puts mail into verbose mode; the details of delivery are displayed on the users terminal. The **−i** flag causes tty interrupt signals to be ignored. This is particularly useful when using *mail* on noisy phone lines. The **−n** flag inhibits the reading of /usr/lib/Mail.rc.

Sending mail. To send a message to one or more people, *mail* can be invoked with arguments which are the names of people to whom the mail will be sent. You are then expected to type in your message, followed by an EOT (control−D) at the beginning of a line. A subject may be specified on the command line by using the **−s** flag. (Only the first argument after the **−s** flag is used as a subject; be careful to quote subjects containing spaces.) The section below, labeled *Replying to or originating mail,* describes some features of *mail* available to help you compose your letter.

Reading mail. In normal usage *mail* is given no arguments and checks your mail out of the post office, then prints out a one line header of each message there. The current message is initially the first message (numbered 1) and can be printed using the **print** command (which can be abbreviated **p**). You can move among the messages much as you move between lines in *ed*, with the commands '+' and '−' moving backwards and forwards, and simple numbers.

Disposing of mail. After examining a message you can **delete** (**d**) the message or **reply** (**r**) to it. Deletion causes the *mail* program to forget about the message. This is not irreversible; the message can be **undeleted** (**u**) by giving its number, or the *mail* session can be aborted by giving the **exit** (**x**) command. Deleted messages will, however, usually disappear never to be 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. 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 command **top** which prints 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** command to set up a response to a message, sending it back to the person who it was from. Text you then type in, up to an end-of-file, defines the contents of the message. While you are composing a message, *mail* treats lines beginning with the character '~' specially. For instance, typing ''~m'' (alone on a line) will place a copy of the current message into the response right shifting it by a tabstop. Other escapes will 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 given in the summary below.)

Ending a mail processing session. You can end a *mail* session with the **quit** (**q**) command. Messages which have been examined go to your *mbox* file unless they have been deleted in which case they are discarded. Unexamined messages go back to the post office. The **−f** option causes *mail* to read in the contents of your *mbox* (or the specified file) for processing; when you **quit**, *mail* writes undeleted messages back to this file. The **−u** flag is a short way of doing "mail **−f** /usr/spool/mail/user".

Personal and systemwide distribution lists. It is also possible to create a 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 jkf mark kridle@ucbcory
```

in the file .mailrc in your home directory. The current list of such aliases can be displayed with the **alias (a)** command in *mail*. System wide distribution lists can be created by editing /usr/lib/aliases, see *aliases*(5) and *sendmail*(8); these are kept in a different syntax. In mail you send, personal aliases will be expanded in mail sent to others so that they will be able to **reply** 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*.

Network mail (ARPA, UUCP, Berknet) See *mailaddr(7)* for a description of network addresses.

Mail has a number of options which can be set in the *.mailrc* file to alter its behavior; thus ''set askcc'' enables the ''askcc'' feature. (These options are summarized below.)

SUMMARY

(Adapted from the 'Mail Reference Manual')

Each command is typed on a line by itself, and may take arguments following the command word. The command need not be typed in its entirety – the first command which matches the typed prefix is used. For commands which take message lists as arguments, if no message list is given, then the next message forward which satisfies the command's requirements is used. 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.

– Goes to the previous message and prints it out. If given a numeric argument *n*, goes to the *n*-th previous message and prints it.

? Prints a brief summary of commands.

! Executes the UNIX shell command which follows.

Print (**P**) Like **print** but also prints out ignored header fields. See also **print** , **ignore** and **retain.**

Reply (**R**) Reply to originator. Does not reply to other recipients of the original message.

Type (**T**) Identical to the **Print** command.

alias (**a**) With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, creates an new or changes an on old alias.

alternates (**alt**) The **alternates** command is useful if you have accounts on several machines. It can be used to inform *mail* that the listed addresses are really you. When you **reply** to messages, *mail* will not send a copy of the message to any of the addresses listed on the *alternates* list. If the **alternates** command is given with no argument, the current set of alternate names is displayed.

chdir (**c**) Changes the user's working directory to that specified, if given. If no directory is given, then changes to the user's login directory.

copy (**co**) The **copy** command does the same thing that **save** does, except that it does not mark the messages it is used on for deletion when you quit.

delete (**d**) Takes a list of messages as argument and marks them all as deleted. Deleted messages will not be saved in *mbox*, nor will they be available for most other commands.

dp (also **dt**) Deletes the current message and prints 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, his *mbox* file, or his edit file in **–f**.

file (**fi**) The same as **folder**.

folders List the names of the folders in your folder directory.

folder (**fo**) The **folder** command switches to a new mail file or folder. With no arguments, it tells you which file you are currently reading. If you give it an argument, it will write out changes (such as deletions) you have made in the current file and read in the new file. Some special conventions are recognized for the name. # means the previous file, % means your system mailbox, %user means user's system mailbox, & means your ~/mbox file, and +folder means a file in your folder directory.

from (**f**) Takes a list of messages and prints their message headers.

headers (**h**) Lists the current range of headers, which is an 18 message group. If a ''+'' argument is given, then the next 18 message group is printed, and if a ''−'' argument is given, the previous 18 message group is printed.

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 **N.B.:** *Ignore* has been superseded by *retain.*
Add the list of header fields named to the *ignored list*. Header fields in the ignore list are not printed on your terminal when you print a message. This command is very handy for suppression of certain machine-generated header fields. The **Type** and **Print** commands can be used to print a message in its entirety, including ignored fields. If **ignore** is executed with no arguments, it lists the current set of ignored fields.

mail (**m**) Takes as argument login names and distribution group names and sends mail to those people.

mbox Indicate that a list of messages be sent to *mbox* in your home directory when you quit. This is the default action for messages if you do *not* have the *hold* option set.

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 (**pre**) A synonym for **hold**.

print (**p**) Takes a message list and types out each message on the user's terminal.

quit (**q**) Terminates the session, saving all undeleted, unsaved messages in the user's *mbox* file in his login directory, preserving all messages marked with **hold** or **preserve** or never referenced in his system mailbox, and removing all other messages from his system mailbox. If new mail has arrived during the session, the message ''You have new mail'' is given. If given while editing a mailbox file with the **−f** flag, then the edit file is rewritten. A return to the Shell is effected, 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 the sender and all recipients of the specified message. The default message must not be deleted.

respond A synonym for **reply**.

retain Add the list of header fields named to the *retained list*. Only the header fields in the retain list are shown on your terminal when you print a message. All other header fields are suppressed. The **Type** and **Print** commands can be used to print a message in its entirety. If **retain** is executed with no arguments, it lists the current set of retained fields.

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'' (no space before or after =) or ''option.''

shell (**sh**) Invokes an interactive version of the shell.

size Takes a message list and prints out the size in characters of each message.

source (**so**) The **source** command reads *mail* commands from a file.

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 message as *not* being deleted.

unread (**U**) Takes a message list and marks each message as *not* having been read.

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**) Similar to **save**, except that *only* the message body (*without* the header) is saved. Extremely useful for such tasks as sending and receiving source program text over the message system.

xit (**x**) A synonym for **exit**.

z *Mail* presents message headers in windowfuls as described under the **headers** command. You can move *mail*'s attention forward to the next window with the **z** command. Also, you can move to the previous window by using **z–**.

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.

~b name ... Add the given names to the list of carbon copy recipients but do not make the names visible in the Cc: line ("blind" carbon copy).

~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 in the current message.

~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 Cause the named string to 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 will be 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.

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. The binary options include the following:

append Causes messages saved in *mbox* to be appended to the end rather than prepended. (This is set in /usr/lib/Mail.rc on version 7 systems.)

ask Causes *mail* to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field will be sent.

askcc Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

autoprint Causes the **delete** command to behave like **dp** – thus, after deleting a message, the next one will be typed automatically.

debug Setting the binary option *debug* is the same as specifying **–d** on the command line and causes *mail* to output all sorts of information useful for debugging *mail*.

dot The binary option *dot* causes *mail* to interpret a period alone on a line as the terminator of a message you are sending.

hold This option is used to hold messages in the system mailbox by default.

ignore Causes interrupt signals from your terminal to be ignored and echoed as @'s.

ignoreeof An option related to *dot* is *ignoreeof* which makes *mail* refuse to accept a control-d as the end of a message. *Ignoreeof* also applies to *mail* command mode.

metoo Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.

nosave Normally, when you abort a message with two RUBOUT, *mail* copies the partial letter to the file ''dead.letter'' in your home directory. Setting the binary option *nosave* prevents this.

Replyall Reverses the sense of *reply* and *Reply* commands.

quiet Suppresses the printing of the version when first invoked.

verbose Setting the option *verbose* is the same as using the **–v** flag on the command line. When mail runs in verbose mode, the actual delivery of messages is displayed on he users terminal.

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.

PAGER Pathname of the program to use in the **more** command or when *crt* variable is set. A default paginator is used if this option is not defined.

SHELL Pathname of the shell to use in the **!** command and the ~! escape. A default shell is used if this option is not defined.

VISUAL	Pathname of the text editor to use in the **visual** command and ~v escape.
crt	The valued option *crt* is used as a threshold to determine how long a message must be before **PAGER** is used to read it.
escape	If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.
folder	The name of the directory to use for storing folders of messages. If this name begins with a '/', *mail* considers it to be an absolute pathname; otherwise, the folder directory is found relative to your home directory.
record	If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not so 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.

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
Message*	temporary for editing messages

SEE ALSO

binmail(1), fmt(1), newaliases(1), aliases(5),
mailaddr(7), sendmail(8)
'The Mail Reference Manual'

BUGS

There are many flags that are not documented here. Most are not useful to the general user.
Usually, *mail* is just a link to *Mail*, which can be confusing.

AUTHOR

Kurt Shoens

NAME

make − maintain program groups

SYNOPSIS

make [**−f** makefile] [option] ... file ...

DESCRIPTION

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **−f** option is present, 'makefile' and 'Makefile' are tried in order. If *makefile* is '−', the standard input is taken. 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 *a*(*b*) means the file named *b* stored in the archive named *a.* A name like *a*((*b*)) means the file stored in archive *a* containing the entry point *b.*

Sharp and newline surround comments.

The following makefile says that 'pgm' depends on two files 'a.o' and 'b.o', and that they in turn depend on '.c' files and a common file 'incl'.

```
pgm: a.o b.o
        cc a.o b.o −lm −o pgm
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:

```
pgm: a.o b.o
        cc a.o b.o −lm −o pgm
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*. In such an entry, the special macro $* stands for the target name with suffix deleted, $@ for the full target name, $< for the complete list of prerequisites, and $? for the list of prerequisites that are out of date. 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**.

Another special macro is 'VPATH'. The 'VPATH' macro should be set to a list of directories separated by colons. When *make* searches for a file as a result of a dependency relation, it will first search the current directory and then each of the directories on the 'VPATH' list. If the file is found, the actual path to the file will be used, rather than just the filename. If 'VPATH' is not defined, then only the current directory is searched.

One use for 'VPATH' is when one has several programs that compile from the same source. The source can be kept in one directory and each set of object files (along with a separate *makefile*) would be in a separate subdirectory. The 'VPATH' macro would point to the source directory in this case.

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'.

Other options:

−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, i.e. 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:'.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1), f77(1), pc(1)
S. I. Feldman *Make − A Program for Maintaining Computer Programs*

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.*

'VPATH' is intended to act like the System V 'VPATH' support, but there is no guarantee that it functions identically.

NAME

man − find manual information by keywords; print out the manual

SYNOPSIS

man [**−**] [**−M** *path*] [section] title ...
man −k keyword ...
man −f file ...

DESCRIPTION

Man is a program which gives information from the programmers manual. It 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 given the option **−k** and a set of keywords, *man* prints out a one line synopsis of each manual sections whose listing in the table of contents contains one of those keywords.

When given the option **−f** and a list of file names, *man* attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither **−k** nor **−f** is specified, *man* formats a specified set of manual pages. If a section specifier is given *man* looks in the that section of the manual for the given *titles. Section* is either an Arabic section number (3 for instance), or one of the words ''new,'' ''local,'' ''old,'' or ''public.'' A section number may be followed by a single letter classifier (for instance, 1g, 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 printing the first section it finds, if any.

If the standard output is a teletype, or if the flag **−** is given, *man* pipes its output through *more*(1) with the option **−s** to crush out useless blank lines and to stop after each page on the screen. Hit a space to continue, a control-D to scroll 11 more lines when the output stops.

Normally *man* checks in a standard location for manual information (/usr/man). This can be changed by supplying a search path (a la the shell) with the **−M** flag. The search path is a colon (':') separated list of directories in which manual subdirectories may be found; e.g. ''/usr/local:/usr/man''. If the environment variable 'MANPATH' is set, its value is used for the default path. If a search path is supplied with the **−k** or **−f** options, it must be specified first.

Man will look for the manual page in either of two forms, the nroff source or preformatted pages. If either version is available, the manual page will be displayed. If the preformatted version is available, and it has a more recent modify time than the nroff source, it will be promptly displayed. Otherwise, the manual page will be formatted with nroff and displayed. If the user has permission, the formatted manual page will be deposited in the proper place, so that later invocations of man will not need to format the page again.

FILES

/usr/man	standard manual area
/usr/man/man?/*	directories containing source for manuals
/usr/man/cat?/*	directories containing preformatted pages
/usr/man/whatis	keyword database

SEE ALSO

apropos(1), more(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* and *talk*(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

mh − Message Handler

SYNOPSIS

any *MH* command

DESCRIPTION

MH is the name of a powerful message handling system. Rather then being a single comprehensive program, *MH* consists of a collection of fairly simple single-purpose programs to send, receive, save, and retrieve messages. The user should refer to the *MH User's Manual* and the pages for the *MH* programs in the Unix Programmers Manual.

Unlike *mail*, the standard UNIX mail user interface program, *MH* is not a closed system which must be explicitly run, then exited when you wish to return to the shell. You may freely intersperse *MH* commands with other shell commands, allowing you to read and answer your mail while you have (for example) a compilation running, or search for a file or run programs as needed to find the answer to someone's question before answering their mail.

The rest of this manual entry is a quick tutorial which will teach you the basics of *MH*. You should read the manual entries for the individual programs for complete documentation.

To get started using *MH*, put the directory **/usr/new/mh** on your **$PATH**. This is best done in one of the files: **.profile**, **.login**, or **.cshrc** in your home directory. (Check the manual entry for the shell you use, in case you don't know how to do this.) Run the *inc* command. If you've never used *MH* before, it will create the necessary default files and directories after asking you if you wish it to do so.

inc moves mail from your system maildrop into your *MH* '+inbox' folder, breaking it up into separate files and converting it to *MH* format as it goes. It prints one line for each message it processes, containing the from field, the subject field and as much of the first line of the message as will fit. It leaves the first message it processes as your current message. You'll need to run *inc* each time you wish to incorporate new mail into your *MH* file.

scan prints a list of the messages in your current folder.

The commands: *show*, *next*, and *prev* are used to read specific messages from the current folder. *show* displays the current message, or a specific message, which may be specified by its number, which you pass as an argument to *show*. *next* and *prev* display, respectively, the message numerically after or before the current message. In all cases, the message displayed becomes the current message. If there is no current message, *show* may be called with an argument, or *next* may be used to advance to the first message.

rmm (remove message) deletes the current message. It may be called with message numbers passed as arguments, to delete specific messages.

repl is used to respond to the current message (by default). It places you in the editor with a prototype response form. While you're in the editor, you may peruse the item you're responding to by reading the file @. After completing your response, type **l** to review it, or **s** to send it.

comp allows you to compose a message by putting you in the editor on a prototype message form, and then lets you send it.

All the *MH* commands may be run with the single argument: '−help', which causes them to print a list of the arguments they may be invoked with.

Commands which take a message number as an argument (*scan*, *show*, *repl*, ...) also take one of the words: *first*, *prev*, *cur*, *next*, or *last* to indicate (respectively) the first, previous, current, next, or last message in the current folder (assuming they are defined).

Commands which take a range of message numbers (*rmm*, *scan*, *show*, ...) also take any of the abbreviations:

<num1>-<num2> - Indicates all messages in the range <num1> to <num2>, inclusive. The range **must** be nonempty.

<num>:+N
<num>:-N - Up to *N* messages beginning with (or ending with) message *num. Num* may be any of the pre-defined symbols: *first, prev, cur, next* or *last.*

first:N
prev:N
next:N
last:N - The first, previous, next or last *N* messages, if they exist.

There are many other possibilities such as creating multiple folders for different topics, and automatically refiling messages according to subject, source, destination, or content. These are beyond the scope of this manual entry.

Following is a list of all the *MH* commands:

ali (1)	– list mail aliases
anno (1)	– annotate messages
burst (1)	– explode digests into messages
comp (1)	– compose a message
dist (1)	– redistribute a message to additional addresses
folder (1)	– set/list current folder/message
folders (1)	– list all folders
forw (1)	– forward messages
inc (1)	– incorporate new mail
mark (1)	– mark messages
mhl (1)	– produce formatted listings of MH messages
mhmail (1)	– send or read mail
mhook (1)	– MH receive−mail hooks
mhpath (1)	– print full pathnames of MH messages and folders
msgchk (1)	– check for messages
msh (1)	– MH shell (and BBoard reader)
next (1)	– show the next message
packf (1)	– compress a folder into a single file
pick (1)	– select messages by content
prev (1)	– show the previous message
prompter (1)	– prompting editor front end
rcvstore (1)	– incorporate new mail asynchronously
refile (1)	– file messages in other folders
repl (1)	– reply to a message
rmf (1)	– remove folder
rmm (1)	– remove messages
scan (1)	– produce a one line per message scan listing
send (1)	– send a message
show (1)	– show (list) messages

sortm (1)	– sort messages
vmh (1)	– visual front–end to MH
whatnow (1)	– prompting front–end for send
whom (1)	– report to whom a message would go
mh–alias (5)	– alias file for MH message system
mh–format (5)	– format file for MH message system
mh–mail (5)	– message format for MH message system
mh–profile (5)	– user customization for MH message system
ap (8)	– parse addresses 822–style
conflict (8)	– search for alias/password conflicts
dp (8)	– parse dates 822–style
install–mh (8)	– initialize the MH environment
post (8)	– deliver a message

FILES

/usr/new/mh	directory containing *MH* commands
/usr/new/lib/mh	*MH* library

SEE ALSO

The Rand MH Message Handling System: User's Manual,
The Rand MH Message Handling System: Tutorial,
The Rand MH Message Handling System: The UCI BBoards Facility,
MH.5: How to process 200 messages a day and still get some real work done

NAME

mkdir − make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777. Standard entries, '**.**', for the directory itself, and **'..'** for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

rmdir(1)

NAME

mkstr − create an error message file by massaging C source

SYNOPSIS

mkstr [−] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process 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 usage of *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, i.e.:

```
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);
}
```

The optional − causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

SEE ALSO

lseek(2), xstr(1)

NAME

more, page − file perusal filter for crt viewing

SYNOPSIS

more [**−cdflsu**] [**−***n*] [**+***linenumber*] [**+/***pattern*] [name ...]

page *more options*

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

The command line options are:

−n An integer which is the size (in lines) of the window which *more* will use instead of the default.

−c *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

−d *More* will prompt the user with the message "Press space to continue, ´q´ to quit." at the end of each screenful, and will respond to subsequent illegal user input by printing "Press ´h´ for instructions." instead of ringing the bell. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

−f This causes *more* to 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 positions, 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 this option is not given, *more* will pause 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 will be cleared before the file is printed.

−s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

−u Normally, *more* will handle 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* will output 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*.

If the program is invoked as *page,* then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed 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* and *msgs* , to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then 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, then 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

i b
: skip back *i* screenfuls and print a screenful of lines

i ^B
: same as b

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), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. 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, then *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, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control−\). *More* will stop sending output, and will display 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.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat,* except that a header is printed before each file (if there is more than one).

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.helpHelp file

SEE ALSO

csh(1), man(1), msgs(1), script(1), sh(1), environ(7)

BUGS

Skipping backwards is too slow on large files.

NAME

mset − retrieve ASCII to IBM 3270 keyboard map

SYNOPSIS

mset

DESCRIPTION

Mset retrieves mapping information for the ASCII keyboard to IBM 3270 terminal special functions. Normally, these mappings are found in */etc/map3270* (see *map3270*(5)). This information is used by the *tn3270* command (see *tn3270*(1)).

Mset can be used store the mapping information in the process environment in order to avoid scanning */etc/map3270* each time *tn3270* is invoked. To do this, place the following command in your *.login* file:

```
set noglob; setenv MAP3270 "`mset`"; unset noglob
```

Mset first determines the user's terminal type from the environment variable **TERM**. Normally *mset* then uses the file */etc/map3270* to find the keyboard mapping for that terminal. However, if the environment variable **MAP3270** exists and contains the entry for the specified terminal, then that definition is used. If the value of **MAP3270** begins with a slash ('/') then it is assumed to be the full pathname of an alternate mapping file and that file is searched first. In any case, if the mapping for the terminal is not found in the environment, nor in an alternate map file, nor in the standard map file, then the same search is performed for an entry for a terminal type of **unknown.** If that search also fails, then a default mapping is used.

FILES

/etc/map3270 keyboard mapping for known terminals

SEE ALSO

tn3270(1), map3270(5)

BUGS

If the entry for the specific terminal exceeds 1024 bytes, *csh*(1) will fail to set the environment variable. *Mset* should probably detect this case and output the path to the *map3270* file instead of the terminal entry.

NAME

msgs − system messages and junk mail program

SYNOPSIS

msgs [**−fhlpq**] [number] [−number]

msgs −s

msgs −c [−days]

DESCRIPTION

Msgs is used to read system messages. These messages are sent by mailing to the login 'msgs' and should be short pieces of information which are suitable to be read once by most users of the system.

Msgs is normally invoked each time you login, by placing it in the file *.login (.profile* if you use */bin/sh*). It will then prompt you with the source and subject of each new message. If there is no subject line, the first few non-blank lines of the message will be displayed. If there is more to the message, you will be told how long it is and asked whether you wish to see the rest of the message. The possible responses are:

y type the rest of the message.

RETURN
synonym for y.

n skip this message and go on to the next message.

− redisplay the last message.

q drops you out of *msgs;* the next time you run the program it will pick up where you left off.

s append the current message to the file ''Messages'' in the current directory; 's−' will save the previously displayed message. A 's' or 's−' may be followed by a space and a file name to receive the message replacing the default ''Messages''.

m or 'm−' causes a copy of the specified message to be placed in a temporary mailbox and *mail*(1) to be invoked on that mailbox. Both 'm' and 's' accept a numeric argument in place of the '−'.

Msgs keeps track of the next message you will see by a number in the file *.msgsrc* in your home directory. In the directory */usr/msgs* it keeps a set of files whose names are the (sequential) numbers of the messages they represent. The file */usr/msgs/bounds* shows the low and high number of the messages in the directory so that *msgs* can quickly determine if there are no messages for you. If the contents of *bounds* is incorrect it can be fixed by removing it; *msgs* will make a new *bounds* file the next time it is run.

The **−s** option is used for setting up the posting of messages. The line

```
msgs: "| /usr/ucb/msgs −s"
```

should be include in */usr/lib/aliases* to enable posting of messages.

The **−c** option is used for performing cleanup on */usr/msgs.* An entry with the **−c** option should be placed in */usr/lib/crontab* to run every night. This will remove all messages over 21 days old. A different expiration may be specified on the command line to override the default.

Options when reading messages include:

−f which causes it not to say ''No new messages.''. This is useful in your *.login* file since this is often the case here.

−q Queries whether there are messages, printing ''There are new messages.'' if there are. The command ''msgs −q'' is often used in login scripts.

−h causes *msgs* to print the first part of messages only.

−l option causes only locally originated messages to be reported.

num A message number can be given on the command line, causing *msgs* to start at the specified message rather than at the next message indicated by your *.msgsrc* file. Thus

    msgs −h 1

prints the first part of all messages.

−number
will cause *msgs* to start *number* messages back from the one indicated by your *.msgsrc* file, useful for reviews of recent messages.

−p causes long messages to be piped through *more*(1).

Within *msgs* you can also go to any specific message by typing its number when *msgs* requests input as to what to do.

FILES

/usr/msgs/*	database
~/.msgsrc	number of next message to be presented

AUTHORS

William Joy
David Wasley

SEE ALSO

aliases(5), crontab(5), mail(1), more(1)

BUGS

NAME

mt − magnetic tape manipulating program

SYNOPSIS

mt [**−f** *tapename*] *command* [*count*]

DESCRIPTION

Mt is used to give 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

SEE ALSO

mtio(4), dd(1), ioctl(2), environ(7)

NAME

mv − move or rename files

SYNOPSIS

mv [**−i**] [**−f**] [**−**] file1 file2

mv [**−i**] [**−f**] [**−**] file ... directory

DESCRIPTION

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; if not, *mv* exits.

In the second form, one or more *files* (plain files or directories) are moved to the *directory* with their original file-names.

Mv refuses to move a file onto itself.

Options:

−i stands for interactive mode. Whenever a move is to supercede an existing file, the user is prompted by the name of the file followed by a question mark. If he answers with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.

−f stands for force. This option overrides any mode restrictions or the −i switch.

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

NAME

netstat − show network status

SYNOPSIS

netstat [**−Aan**] [**−f** *address_family*] [*system*] [*core*]
netstat [**−himnrs**] [**−f** *address_family*] [*system*] [*core*]
netstat [**−n**] [**−I** *interface*] *interval* [*system*] [*core*]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with an *interval* specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces.

The options have the following meaning:

−A With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

−a With the default display, show the state of all sockets; normally sockets used by server processes are not shown.

−h Show the state of the IMP host table.

−i Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown).

−I *interface*
Show information only about this interface; used with an *interval* as described below.

−m Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).

−n Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.

−s Show per-protocol statistics.

−r Show the routing tables. When **−s** is also present, show routing statistics instead.

−f *address_family*
Limit statistics or address control block reports to those of the specified *address family*. The following address families are recognized: *inet*, for **AF_INET**, *ns*, for **AF_NS**, and *unix*, for **AF_UNIX**.

The arguments, *system* and *core* allow substitutes for the defaults ''/vmunix'' and ''/dev/kmem''.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form ''host.port'' or ''network.port'' if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **−n** option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet ''dot format,'' refer to *inet*(3N). Unspecified, or ''wildcard'', addresses and ports appear as ''*''.

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit (''mtu'') are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route (''U'' if ''up''), whether the route is to a gateway (''G''), and whether the route was created dynamically by a redirect (''D''). Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the *−I* option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

iostat(1), vmstat(1), hosts(5), networks(5), protocols(5), services(5), trpt(8C)

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

NAME

newaliases − rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

Newaliases rebuilds the random access data base for the mail aliases file */usr/lib/aliases*. It must be run each time */usr/lib/aliases* is changed in order for the change to take effect.

SEE ALSO

aliases(5), sendmail(8)

NAME

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.

FILES

nohup.out standard output and standard error file under *nohup*

SEE ALSO

csh(1), setpriority(2), 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 [**−agnopru**] [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, or **−** for debugger 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 are:

−a Print symbol table entries inserted for use by debuggers.

−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.

SEE ALSO

ar(1), ar(5), a.out(5), stab(5)

NAME

notes, autoseq, readnotes − a news system

SYNOPSIS

notes [**−sxin**] [**-o** *date-spec*] [**-a** subsequencer] [**−t** termtype] [**-f** file] topic1 [...]

autoseq [**-a** subsequencer]

readnotes [**-a** subsequencer]

DESCRIPTION

Notes supports computer managed discussion forums. It coordinates access to and updates of data bases of notes and their responses. A single *notesfile* contains an ordered list of *base notes,* each of which may have an ordered list of responses associated with it. A *note string* consists of a *base note* and all of its responses. Separate *notesfiles* contain discussions on separate subject matters; microcomputers might be discussed in a ''micronotes'' notesfile while bicycling enthusiasts make their comments in a ''bicycle'' notesfile.

The **−s** option signals *notes* to use the automatic sequencer. With the sequencer enabled, *notes* shows the new notes and responses since your last entry into that notesfile. With the sequencer enabled by **−s** the notes program will not enter notesfiles which have no new text. Specify **−x** to use the sequencer and enter notesfiles even if they have no new text. The **−i** and **−n** options are still more sequencing modes, **−i** is similar to **−s** but shows the index page instead of the first modified note. **−n** turns the sequencer off.

The **-o** option helps users find articles that are vaguely remembered with terms such as ''some note in the last 3 days in one of several notesfiles''. These can be found with a command like:

```
notes -o "3 days ago" nf1 nf2 nf3
```

which sequences past all articles written in notesfiles ''nf1'', ''nf2'', and ''nf3'' in the last 3 days. The user's timestamps are not updated.

The **-a** option specifies a *subsequencer*. This allows several people sharing the same signon to maintain their own sequencer file. The actual sequencer name is generated by concatenating the user name and the subsequencer name. It is recommended that subsequencer names be unique within the first 6 characters.

Specify **−t** *termtype* to override the TERM environment variable. This switch is primarily for V6 systems.

The **-f** option directs *notes* to read the contents of a file for a list of notesfiles to scan. This file and the directories must be readable by the notesfile user id.

The topic list specifies which notesfiles are to be scanned. The notesfiles are scanned from left to right; upon finishing the first topic, the second is entered. The shell's meta-characters are recognized within a topic but must be escaped to prevent shell interpretation. Specifying ''net.*'' will yield all the notesfiles with the prefix ''net.''. Specify ''*unix*'' to read all notesfiles with the string ''unix'' in their names. Bracket and question mark constructs are also recognized.

Notesfile names are parsed such that a notesfile will be entered only once no matter how many times it is listed on the command line and in any files specified by the **-f** option. Notesfiles can also be excluded by prefixing their names with a '!'. Thus to see all notesfiles except ''general'', one might type:

```
notes "*" !general
```

The *autoseq* and *readnotes* commands allow sequencing through a list of notesfiles with the sequencer enabled using with a single command. *Autoseq* and *Readnotes* function identically. They are syntactically equivalent to ''notes -s $NFSEQ''. The environment variable NFSEQ contains a comma separated list of notesfile specifications. A typical *NFSEQ* definition for the Bourne shell looks like:

NFSEQ=‘‘general,announce,net.*,bicycle,srg,:/usr/essick/nflist’’.

Specifications beginning with a ‘:’ specify a file to read for more notesfile names. In the previous example, the last specification reads the contents of the file ‘/usr/essick/nflist’ for more notesfile specifications. Many of these can appear in the *NFSEQ* variable.

Notes and responses are entered by using an editor. The default editor is *ed*(1). This can be changed by setting one of the environment variables *NFED* or *EDITOR. Notes* looks for *NFED* before looking for *EDITOR,* allowing users to use different editors for writing notes and for other tools.

Some commonly used commands within the notesfile system are listed below:

space Show the next page of the note/response.

; Go the next response, if there are no more responses go to the next note.

- Go to the previous page of the current note/response. From the first page of a response, go to the previous response (or the base note from the first response). From the first page of a base note, go to the previous note.

newline Go to the next note.

j Jump to the next unread note/response (when using sequencer).

J Jump to the next unread note, ignoring any further responses in the current note string (when using sequencer).

w When issued from the index page enters a new note. When entered from a note/response display enters a response. A capital-W will include the text of the currently displayed note/response in the new response.

q Leave the current notesfile.

Q Leave the current notesfile without updating the sequencer information.

control-d Return to the shell, ignoring any further notesfiles in the current invocation. No sequencer information is updated.

x Search for a note with the (prompted for) string in its title. Capital-X asks for a new search string, otherwise the last entered string is used.

s Saves the currently displayed note/response at the end of a (prompted for) file. Capital-S saves the entire note string.

M Sends the text of the note/response displayed and your comments to another user(s). The P command routes the letter to the author of the note/response.

t Issues a write(1) command to the author of the currently displayed note/response. No action is taken if the note originated on a remote system or is anonymous.

! Forks a shell.

Only the *notesfile owner* can create new notesfiles. The *notesfile owner* will create the notesfile and turn control over to the person requesting the notesfile. This person is the *notesfile director;* he may designate others to also be *notesfile directors.* The *notesfile director* has special privileges including: deleting any note, determining policy for the notesfile, permitting anonymous notes, and determining accessibility of the notesfile.

An interface is provided to *news*(1). Transfers in both directions are supported. See *newsoutput*(8) and the *Notesfile Reference Manual* for more information on this facility.

Facilities for mailing to notesfiles (*nfmail*(8)), networking notesfiles (*nfxmit*(8)), printing notesfiles (*nfprint*(1)), archiving old notes (*nfarchive*(8)), and several user routines (*nfabort*(3) and *nfcomment*(3)) exist.

The concept of a notesfile was taken from the PLATO system (a trademark of Control Data Corporation) designed at the University of Illinois to provide automated teaching capabilities.

FILES

/etc/passwd	for the users name
/etc/group	for the users group(s)
/etc/termcap	for terminal capabilites
/usr/spool/notes	the default notesfile data base
/usr/spool/notes/.utilities	utility programs and online help
/usr/spool/notes/.sequencer/*user*	Sequencing timestamps for *user.*
/usr/spool/notes/.sequencer/*user*:*subsequencer*	Sub-sequencing timestamps for *user.*

SEE ALSO

ed(1), news(1N), termcap(3), write(1), checknotes(1N), mknf(8N), newsoutput(8N), nfabort(3N), nfaccess(8N), nfarchive(8N), nfmail(8N), nfpipe(1N), nfprint(1N), nfstats(1N), nfxmit(8N), nfcomment(3N), notes(8N),
(many of which are included in (*The Notesfile Reference Manual (USD:11)*)

AUTHORS

Ray Essick (uiucdcs!essick, essick%uiuc@csnet-relay.arpa)
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL

Rob Kolstad (kolstad@convex.UUCP)
CONVEX Computer Corporation
Richardson, TX

NAME

nroff − text formatting

SYNOPSIS

nroff [option] ... [file] ...

DESCRIPTION

Nroff formats text in the named *files* for typewriter-like devices. See also *troff(1).* The full capabilities of *nroff* are described in the *Nroff/Troff User's Manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (**−**) is taken to be a file name corresponding to the standard input.

The options, which may appear in any order so long as they appear *before* the files, are:

−o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end.

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.

−m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

−T*name* Prepare output for specified terminal. Known *names* are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).

−e Produce equally-spaced words in adjusted lines, using full terminal resolution.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

J. F. Ossanna, *Nroff/Troff user's manual*
B. W. Kernighan, *A TROFF Tutorial*
troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

NAME

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** 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 ACSII names. If the **p** character is given also, then bytes with even parity are underlined. The **P** character causes bytes with odd parity to be 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

chfn, chsh, passwd − change password file information

SYNOPSIS

passwd [**−f**] [**−s**] [name]

DESCRIPTION

This command changes (or installs) a password, login shell (**−s** option), or GECOS information field (**−f** option) associated with the user *name* (your own name by default).

When altering a password, the program prompts for the current 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.

When altering a login shell, *passwd* displays the current login shell and then prompts for the new one. The new login shell must be one of the approved shells listed in */etc/shells* unless you are the super-user. If */etc/shells* does not exist, the only shells that may be specified are */bin/sh* and */bin/csh*.

The super-user may change anyone's login shell; normal users may only change their own login shell.

When altering the GECOS information field, *passwd* displays the current information, broken into fields, as interpreted by the *finger*(1) program, among others, and prompts for new values. These fields include a user's ''real life'' name, office room number, office phone number, and home phone number. Included in each prompt is a default value, which is enclosed between brackets. The default value is accepted simply by typing a carriage return. To enter a blank field, the word ''none'' may be typed. Below is a sample run:

> **Name [Biff Studsworth II]:**
> **Room number (Exs: 597E or 197C) []:** 521E
> **Office Phone (Ex: 1632) []:** 1863
> **Home Phone (Ex: 987532) [5771546]:** none

Passwd allows phone numbers to be entered with or without hyphens. It is a good idea to run *finger* after changing the GECOS information to make sure everything is setup properly.

The super-user may change anyone's GECOS information; normal users may only change their own.

FILES

/etc/passwd	The file containing all of this information
/etc/shells	The list of approved shells

SEE ALSO

login(1), finger(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, *UNIX password security*

NAME

patch − a program for applying a diff file to an original

SYNOPSIS

patch [options] orig diff [+ [options] orig]

DESCRIPTION

Patch will take a patch file containing any of the three forms of difference listing produced by the *diff* program and apply those differences to an original file, producing a patched version. By default, the patched version is put in place of the original, with the original file backed up to the same name with the extension ''.orig'', or as specified by the **-b** switch. You may also specify where you want the output to go with a **-o** switch. If *diff* is omitted, or is a hyphen, the patch will be read from standard input.

Upon startup, patch will attempt to determine the type of the diff file, unless over-ruled by a **-c**, **-e**, or **-n** switch. Context diffs and normal diffs are applied by the *patch* program itself, while ed diffs are simply fed to the *ed* editor via a pipe.

Patch will try to skip any leading garbage, apply the diff, and then skip any trailing garbage. Thus you could feed an article or message containing a context or normal diff to *patch*, and it should work. If the entire diff is indented by a consistent amount, this will be taken into account.

With context diffs, and to a lesser extent with normal diffs, *patch* can detect when the line numbers mentioned in the patch are incorrect, and will attempt to find the correct place to apply each hunk of the patch. As a first guess, it takes the line number mentioned for the hunk, plus or minus any offset used in applying the previous hunk. If that is not the correct place, *patch* will scan both forwards and backwards for a set of lines matching the context given in the hunk. All lines of the context must match. If *patch* cannot find a place to install that hunk of the patch, it will put the hunk out to a reject file, which normally is the name of the output file plus ''.rej''. (Note that the rejected hunk will come out in context diff form whether the input patch was a context diff or a normal diff. If the input was a normal diff, many of the contexts will simply be null.)

If no original file is specified on the command line, *patch* will try to figure out from the leading garbage what the name of the file to edit is. In the header of a context diff, the filename is found from lines beginning with ''***'' or ''---'', with the shortest name of an existing file winning. Only context diffs have lines like that, but if there is an ''Index:'' line in the leading garbage, *patch* will try to use the filename from that line. The context diff header takes precedence over an Index line. If no filename can be intuited from the leading garbage, you will be asked for the name of the file to patch.

(If the original file cannot be found, but a suitable SCCS or RCS file is handy, *patch* will attempt to get or check out the file.)

Additionally, if the leading garbage contains a ''Prereq: '' line, *patch* will take the first word from the prerequisites line (normally a version number) and check the input file to see if that word can be found. If not, *patch* will ask for confirmation before proceeding.

The upshot of all this is that you should be able to say, while in a news interface, the following:

    | patch -d /usr/src/local/blurfl

and patch a file in the blurfl directory directly from the article containing the patch.

If the patch file contains more than one patch, *patch* will try to apply each of them as if they came from separate patch files. This means, among other things, that it is assumed that separate patches will apply to separate files, and that the garbage before each patch will be examined for interesting things such as filenames and revision level, as mentioned previously. You can give switches (and another original file name) for the second and subsequent patches by separating the corresponding argument lists by a '+'. The argument list for a second or subsequent patch may not specify a new patch file, however.

Patch recognizes the following switches:

−b causes the next argument to be interpreted as the backup extension, to be used in place of ''.orig''.

−c forces *patch* to interpret the patch file as a context diff.

−d causes *patch* to interpret the next argument as a directory, and cd to it before doing anything else.

−D causes *patch* to use the "#ifdef...#endif" construct to mark changes. The argument following will be used as the differentiating symbol. Note that, unlike the C compiler, there must be a space between the **−D** and the argument.

−e forces *patch* to interpret the patch file as an ed script.

−l causes the pattern matching to be done loosely, in case the tabs and spaces have been munged in you input file. Any sequence of whitespace in the pattern line will match any sequence in the input file. Normal characters must still match exactly. Each line of the context must still match a line in the input file.

−n forces *patch* to interpret the patch file as a normal diff.

−N forces patch to not try and reverse the diffs if it thinks that they may have been swapped. See the **−R** option below.

−o causes the next argument to be interpreted as the output file name.

−p causes leading pathnames to be kept. If the diff is of the file ''b/a.c'', patch will look for ''a.c'' in the ''b'' directory, instead of the current directory. This probably won't work if the diff has rooted pathnames.

−r causes the next argument to be interpreted as the reject file name.

−R tells *patch* that this patch was created with the old and new files swapped. (Yes, I'm afraid that does happen occasionally, human nature being what it is.) *Patch* will attempt to swap each hunk around before applying it. Rejects will come out in the swapped format. The **−R** switch will not work with ed diff scripts because there is too little information to reconstruct the reverse operation.

If the first hunk of a patch fails, *patch* will reverse the hunk to see if it can be applied that way unless the **−N** option is supplied. If it can, the **−R** switch will be set automatically. If it can't, the patch will continue to be applied normally. (Note: this method cannot detect a reversed patch if it is a normal diff and if the first command is an append (i.e. it should have been a delete) since appends always succeed. Luckily, most patches add lines rather than delete them, so most reversed normal diffs will begin with a delete, which will fail, triggering the heuristic.)

−s makes *patch* do its work silently, unless an error occurs.

−x<number>
sets internal debugging flags, and is of interest only to *patch* patchers.

ENVIRONMENT

No environment variables are used by *patch*.

FILES

/tmp/patch*

SEE ALSO

diff(1)

DIAGNOSTICS

Too many to list here, but generally indicative that *patch* couldn't parse your patch file.

The message ''Hmm...'' indicates that there is unprocessed text in the patch file and that *patch* is attempting to intuit whether there is a patch in that text and, if so, what kind of patch it is.

CAVEATS

Patch cannot tell if the line numbers are off in an ed script, and can only detect bad line numbers in a normal diff when it finds a ''change'' command. Until a suitable interactive interface is added, you should probably do a context diff in these cases to see if the changes made sense. Of course, compiling without errors is a pretty good indication that it worked, but not always.

Patch usually produces the correct results, even when it has to do a lot of guessing. However, the results are guaranteed to be correct only when the patch is applied to exactly the same version of the file that the patch was generated from.

BUGS

Could be smarter about partial matches, excessively deviant offsets and swapped code, but that would take an extra pass.

If code has been duplicated (for instance with #ifdef OLDCODE ... #else ... #endif), *patch* is incapable of patching both versions, and, if it works at all, will likely patch the wrong one, and tell you it succeeded to boot.

If you apply a patch you've already applied, *patch* will think it is a reversed patch, and un-apply the patch. This could be construed as a feature.

NAME

pc − Pascal compiler

SYNOPSIS

pc [option] [**−i** name ...] name ...

DESCRIPTION

Pc is a Pascal compiler. If given an argument file ending with **.p**, it will compile the file and load 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 **function**s and **procedure**s 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 **function**s and **procedure**s 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 Berkeley Pascal User's Manual for details.

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 Have the compiler 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 suffixed '.s'. (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** calls, and initialize all variables to zero as in *pi*.

−b Block buffer the file *output.*

−i 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.

−t directory
Use the given *directory* for compiler temporary files.

−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 *Berkeley Pascal User's 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.*strings	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

Berkeley Pascal User's Manual
pi(1), pxp(1), pxref(1), sdb(1)

DIAGNOSTICS

For a basic explanation do

pc

See *pi*(1). for an explanation of the error message format. Internal errors cause messages containing the word SNARK.

AUTHORS

Charles B. Haley, William N. Joy, and Ken Thompson
Retargetted to the second pass of the portable *C* compiler by Peter Kessler
Runtime library and inline optimizer by M. Kirk McKusick
Separate compilation consistency checking by Louise Madrid

BUGS

The keyword **packed** is recognized but has no effect.

The binder is not as strict as described here, with regard to the rules about external declarations only in '.h' files and including '.h' files only at the outermost level. It will be made to perform these checks in its next incarnation, so users are warned not to be sloppy.

The **−z** flag doesn't work for separately compiled files.

Because the **−s** option is usurped by the compiler, it is not possible to pass the strip option to the loader. Thus programs which are to be stripped, must be run through *strip*(1) after they are compiled.

NAME

pdx − pascal debugger

SYNOPSIS

pdx [**−r**] [*objfile*]

DESCRIPTION

Pdx is a tool for source level debugging and execution of Pascal programs. The *objfile* is an object file produced by the Pascal translator *pi*(1). If no *objfile* is specified, *pdx* looks for a file named ''obj'' in the current directory. The object file contains a symbol table which includes the name of the all the source files translated by *pi* to create it. These files are available for perusal while using the debugger.

If the file ''.pdxinit'' exists in the current directory, then the debugger commands in it are executed.

The **−r** option causes the *objfile* to be executed immediately; if it terminates successfully *pdx* exits. Otherwise it reports the reason for termination and offers the user the option of entering the debugger or simply letting *px* continue with a traceback. If **−r** is not specified, *pdx* just prompts and waits for a command.

The commands are:

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.

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

Have tracing information printed when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

The first argument describes what is to be traced. If it is a *source-line-number*, then the line is printed immediately prior to being executed. Source line numbers in a file other than the current one must be preceded by the name of the file and a colon, e.g. ''mumble.p:17''.

If the argument is a procedure or function name then every time it is called, information is printed 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 then the value it is returning is also printed.

If the argument is an *expression* with an **at** clause then the value of the expression is printed whenever the identified source line is reached.

If the argument is a variable then the name and value of the variable is printed whenever it changes. Execution is substantially slower during this form of tracing.

If no argument is specified then all source lines are printed before they are executed. Execution is substantially slower during this form of tracing.

The clause ''**in** *procedure/function*'' restricts tracing information to be printed only while executing inside the given procedure or function.

Condition is a Pascal boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

There is no restriction on the amount of information that can be traced.

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.

delete *command-number*
The trace or stop corresponding to the given number is removed. The numbers associated with traces and stops are printed by the **status** command.

status [**>** *filename*]
Print out the currently active **trace** and **stop** commands.

cont Continue execution from where it stopped. This can only be done when the program was stopped by an interrupt or through use of the **stop** command.

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 the **step** command will stop at the beginning of that block, while the **next** command will not.

print *expression* [**,** *expression* ...]
Print out the values of the Pascal expressions. Variables declared in an outer block but having the same identifier as one in the current block may be referenced as ''*block-name* **.** *variable*''.

whatis *identifier*
Print the declaration of the given identifier.

which *identifier*
Print the full qualification of the given identifer, i.e. the outer blocks that the identifier is associated with.

assign *variable expression*
Assign the value of the expression to the variable.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function.

help Print out a synopsis of *pdx* commands.

gripe Invokes a mail program to send a message to the person in charge of *pdx*.

where Print out a list of the active procedures and functions and the respective source line where they are called.

source *filename*
Read *pdx* commands from the given *filename*. Especially useful when the *filename* has been created by redirecting a **status** command from an earlier debugging session.

dump [**>** *filename*]
Print the names and values of all active data.

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. As in the editor ''$'' can be used to refer to the last line. If no lines are specified, the entire file is 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.

file [*filename*]
Change the current source file name to *filename*. If none is specified then the current source file name is printed.

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.

pi Recompile the program and read in the new symbol table information.

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*
This command makes *pdx* respond to *new-command-name* the way it used to respond to *old-command-name*.

quit Exit *pdx*.

The following commands deal with the program at the *px* instruction level rather than source level. They are not intended for general use.

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*address*] [**if** *cond*]
stopi [**at**] [*address*] [**if** *cond*]
Turn on tracing or set a stop using a *px* machine instruction addresses.

xi *address* [**,** *address*]
Print the instructions starting at the first *address*. Instructions up to the second *address* are printed.

xd *address* [**,** *address*]
Print in octal the specified data location(s).

FILES

obj	Pascal object file
.pdxinit	*Pdx* initialization file

SEE ALSO

pi(1), px(1)
An Introduction to Pdx

BUGS

Pdx does not understand sets, and provides no information about files.

The *whatis* command doesn't quite work for variant records.

Bad things will happen if a procedure invoked with the **call** command does a non-local goto.

The commands **step** and **next** should be able to take a *count* that specifies how many lines to execute.

There should be commands **stepi** and **nexti** that correspond to **step** and **next** but work at the instruction level.

There should be a way to get an address associated with a line number, procedure or function, and variable.

Most of the command names are too long.

The alias facility is quite weak.

A *csh*-like history capability would improve the situation.

NAME

pi − Pascal interpreter code translator

SYNOPSIS

pi [**option**] [**−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.

The following flags are interpreted by *pi;* the associated options can also be controlled in comments within the program as described in the *Berkeley Pascal User's 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'.

AUTHORS

Charles B. Haley, William N. Joy, and Ken Thompson
Ported to VAX-11 by Peter Kessler

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**] [**−i** 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/ucb/pi	Pascal translator
/usr/ucb/px	Pascal executor
/tmp/pix*	temporary
/usr/lib/how_pix	basic explanation

SEE ALSO

Berkeley Pascal User's Manual
pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

pix

NAME

plot − graphics filters

SYNOPSIS

plot [**−T**terminal] [**−r**resolution] [files...]

DESCRIPTION

These commands read plotting instructions (see *plot*(5)) from the standard input or the specified *files*, and in general produce plotting instructions suitable for a particular *terminal* on the standard output. The **−r** flag may be used to specify the device's output resolution (currently only the Imagen laser printer understands this option).

If no *terminal* type is specified, the environment parameter $TERM (see *environ*(7)) is used. Known *terminals* are:

4013 Tektronix 4013 storage scope.

4014 or **tek**
Tektronix 4014 or 4015 storage scope with Enhanced Graphics Module. (Use 4013 for Tektronix 4014 or 4015 without the Enhanced Graphics Module).

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

aed AED 512 color graphics terminal.

bitgraph or **bg**
BBN bitgraph graphics terminal.

imagen or ip
Imagen laser printer (default 240 dots-per-inch resolution).

crt Any crt terminal capable of running *vi*(1).

dumb Dumb terminals without cursor addressing or line printers.

vt125 DEC vt125 terminal.

hp2648 or **hp** or **hp8**
Hewlett Packard 2648 graphics terminal.

ver Versatec D1200A printer-plotter.

var Benson Varian printer-plotter.

These versions of *plot* use the **−g** option of *lpr*(1) to send the result directly to the plotter device rather than to the standard output.

FILES

/usr/bin/t4013
/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/aedplot
/usr/bin/bgplot
/usr/bin/crtplot
/usr/bin/dumbplot
/usr/bin/gigiplot
/usr/bin/hpplot
/usr/bin/implot
/usr/ucb/lpr

SEE ALSO

plot(3X), plot(3F), plot(5), lpr(1)

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 *Berkeley Pascal User's Manual.*

AUTHOR

M. Kirk McKusick

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

pr − print file

SYNOPSIS

pr [option] ... [file] ...

DESCRIPTION

Pr produces a printed listing of one or more *files.* The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, *pr* prints its standard input.

Options apply to all following files but may be reset between files:

−*n* Produce *n*-column output.

+*n* Begin printing with page *n.*

−h Take the next argument as a page 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.

−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 *files* simultaneously, each in one column,

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(1)

DIAGNOSTICS

There are no diagnostics when *pr* is printing on a terminal.

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(7), csh(1)

NAME

prof − display profile data

SYNOPSIS

prof [**−a**] [**−l**] [**−n**] [**−z**] [**−s**] [**−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* 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.

In order for the number of calls to a routine to be tallied, the **−p** option of *cc, f77* or *pc* must have been given when the file containing the routine was compiled. This option also arranges for the profile file to be produced automatically.

Options are:

−a all symbols are reported rather than just external symbols.

−l the output is sorted by symbol value.

−n the output is sorted by number of calls

−s a summary profile file is produced in *mon.sum.* This is really only useful when more than one profile file is specified.

−v all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the *plot*(1) filters. When plotting, the numbers *low* and *high,* by default 0 and 100, may be given to cause a selected percentage of the profile to be plotted with accordingly higher resolution.

−z routines which have zero usage (as indicated by call counts and accumulated time) are nevertheless printed in the output.

FILES

mon.out for profile
a.out for namelist
mon.sum for summary profile

SEE ALSO

monitor(3), profil(2), cc(1), plot(1G)

BUGS

Beware of quantization errors.

Is confused by *f77* which puts the entry points at the bottom of subroutines and functions.

NAME

ps − process status

SYNOPSIS

ps [**acegklnstuvwxU#**]

DESCRIPTION

Ps prints information about processes. Normally, only your processes are candidates to be printed by *ps;* specifying **a** causes other users' processes to be candidates to be printed; 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, e.g. ''RWNA''. The first letter indicates the runnability of the process: R for runnable processes, T for stopped processes, P for processes in page wait, D for those in disk (or other short term) waits, S for those sleeping for less than about 20 seconds, and I for idle (sleeping longer than about 20 seconds) processes. The second letter indicates whether a process is swapped out, showing W if it is, or a blank if it is loaded (in-core); a process which has specified a soft limit on memory requirements and which is exceeding that limit shows >; such a process is (necessarily) not swapped. The third letter indicates whether a process is running with altered CPU scheduling priority (nice); if the process priority is reduced, an N is shown, if the process priority has been artificially raised then a '<' is shown; processes running without special treatment have just a blank. The final letter indicates any special treatment of the process for virtual memory replacement; the letters correspond to options to the *vadvise*(2) call; currently the possibilities are A standing for VA_ANOM, S for VA_SEQL and blank for VA_NORM; an A typically represents a *lisp*(1) in garbage collection, S is typical of large image processing programs which are using virtual memory to sequentially address voluminous data.

Here are the options:

a asks for information about all processes with terminals (ordinarily only one's own processes are displayed).

c prints 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 Asks for the environment to be printed as well as the arguments to the command.

g Asks for 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 causes the file */vmcore* is used in place of */dev/kmem* and */dev/mem.* This is used for postmortem system debugging.

l asks for a long listing, with fields PPID, CP, PRI, NI, ADDR, SIZE, RSS and WCHAN as described below.

n Asks for numerical output. In a long listing, the WCHAN field is printed numerically rather than symbolically, or, in a user listing, the USER field is replaced by a UID field.

s Adds the size SSIZ of the kernel stack of each process (for use by system maintainers) to the basic output format.

t*x* restricts output to processes whose controlling tty is *x* (which should be specified as printed by *ps,* e.g. *t3* for tty3, *tco* for console, *td0* for ttyd0, *t?* for processes with no tty, *t* for processes at the current tty, etc). This option must be the last one given.

u A user oriented output is produced. This includes fields USER, %CPU, NICE, SIZE, and RSS as described below.

v A version of the output containing virtual memory statistics is output. 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, e.g. ww, use arbitrarily wide output. This information is used to decide how much of long commands to print.

x asks even about processes with no terminal.

U causes ps to update a private database where is keeps system information. Thus ''ps U'' should be included in the /etc/rc file.

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 is taken to be the file containing the system's namelist. Otherwise, /vmunix is used. A third argument tells *ps* where to look for *core* if the **k** option is given, instead of /vmcore. If a fourth argument is given, it is taken to be the name of a swap file to use instead of the default /dev/drum.

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))
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 *setrlimit*(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). A symbol is chosen that classifies the address, unless numerical output is requested (see the **n** flag). In this case, the initial part of the address is trimmed off and is printed hexadecimally, e.g., 0x80004000 prints as 4000.

F flags associated with process as in <*sys/proc.h*>:

SLOAD	000001	in core
SSYS	000002	swapper or pager process
SLOCK	000004	process being swapped out
SSWAP	000008	save area flag
STRC	000010	process is being traced
SWTED	000020	another tracing flag
SULOCK	000040	user settable lock in core
SPAGE	000080	process in page wait state
SKEEP	000100	another flag to prevent swap out
SDLYU	000200	delayed unlock of pages
SWEXIT	000400	working on exiting
SPHYSIO	000800	doing physical i/o (bio.c)

SVFORK	001000	process resulted from vfork()
SVFDONE	002000	another vfork flag
SNOVM	004000	no vm, parent in a vfork()
SPAGI	008000	init data space on demand from inode
SANOM	010000	system detected anomalous vm behavior
SUANOM	020000	user warned of anomalous vm behavior
STIMO	040000	timing out during sleep
SDETACH	080000	detached inherited by init
SOUSIG	100000	using old signal mechanism

A process that has exited and has a parent that has not yet waited for the process 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
/etc/psdatabase	system namelist, device, and wait channel information

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

ptx − permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an *nroff* or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head,* at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter; the default line length is 100 characters.

−w *n* Use the next argument, *n,* as the width of the output line. The default line length is 72 characters.

−g *n* Use the next argument, *n,* as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

−o only Use as keywords only the words given in the *only* file.

−i ignore
Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use */usr/lib/eign* as the *ignore* file.

−b break
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

−r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx.*

FILES

/usr/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

pwd − working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

cd(1), csh(1), getwd(3)

BUGS

In *csh*(1) the command *dirs* is always faster (although it 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).

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

Berkeley Pascal User's 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 an internal inconsistency detected in *px* probably due to a Pascal system bug.

AUTHORS

Charles B. Haley, William Joy, and Ken Thompson
VAX-11 version by Kirk McKusick

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,* to execute the program, and to then issue the command

pxp −z name.p

A reformatted listing is output if none of the **c**, **t**, or **z** options are specified; thus

pxp old.p > new.p

places a pretty-printed version of the program in 'old.p' in the file 'new.p'.

The use of the following options of *pxp* is discussed in sections 2.6, 5.4, 5.5 and 5.10 of the *Berkeley Pascal User's 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 *name* s, are given the profile is of the entire program. If a list of names is given, then only any specified **procedure**s or **function**s and the contents of any specified **include** files will appear in the profile.

−_ Underline keywords.

−*d* With *d* a digit, $2 \le d \le 9$, causes *pxp* to use *d* spaces 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

Berkeley Pascal User's 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.

AUTHOR

William Joy

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

Berkeley Pascal User's Manual

AUTHOR

Niklaus Wirth

BUGS

Identifiers are trimmed to 10 characters.

NAME

quota − display disc usage and limits

SYNOPSIS

quota [**−qv**] [user]

DESCRIPTION

Quota displays users' disc usage and limits. Only the super-user may use the optional *user* argument to view the limits of users other than himself.

The **−q** flag prints a more terse message, containing only information on file systems where usage is over quota.

If a **−v** flag is supplied, *quota* will also display user's quotas on file systems where no storage is allocated.

Quota reports only on file systems which have disc quotas. If *quota* exits with a non-zero status, one or more file systems are over quota.

SEE ALSO

quota(2), quotaon(8)

NAME

ranlib − convert archives to random libraries

SYNOPSIS

ranlib [**−t**] 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.

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

SEE ALSO

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

BUGS

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

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

rcp − remote file copy

SYNOPSIS

rcp [**−p**] file1 file2
rcp [**−p**] [**−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** option 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.

By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the *umask*(2) on the destination host is used. The **−p** option causes *rcp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the *umask*.

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 ''rname@rhost'' to use *rname* rather than the current user name on the remote host. The destination hostname may also take the form ''rhost.rname'' to support destination machines that are running 4.2BSD versions of *rcp*.

SEE ALSO

cp(1), 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.

NAME

rcsintro − introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of text files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, form letters, etc.

The basic user interface is extremely simple. The novice only needs to learn two commands: *ci* and *co*. *Ci*, short for "checkin", deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. *Co*, short for "checkout", retrieves revisions from an RCS file.

SEE ALSO

ci(1), co(1), ident(1), merge(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(5). all in
Walter F. Tichy, ''An Introduction to the Revision Control System'', Programmer Supplementary Documents, Volume 1 (PS1:13)

NAME

rdist − remote file distribution program

SYNOPSIS

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

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

DESCRIPTION

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

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

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

Other options:

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

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

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

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

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

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

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

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

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

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

−b Binary comparison. Perform a binary comparison and update files if they differ rather than comparing dates and sizes.

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

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

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

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

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

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

The source and destination lists have the following format:

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

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

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

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

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

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

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

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

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

The following is a small example.

```
HOSTS = ( matisse root@arpa)

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

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

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

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

IMAGEN = (ips dviimp catdvi)

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

BUGS

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

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

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

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

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

NAME

readnews − read news articles

SYNOPSIS

readnews [**−a** *date*] [**−n** *newsgroups*] [**−t** *titles*] [**−leprxhfuM**] [**−c** [*mailer*]]

readnews −s

DESCRIPTION

Readnews without argument prints unread articles. There are several interfaces available other than the default:

Flag	Interface
−M	An interface to *mailx*(1).
−c	A *binmail*(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.
−p	All selected articles are sent to the standard output. No questions asked.
−l	Only the titles output. The *.newsrc* file will not be updated.
−e	Like **−l** but also updates the *.newsrc* file.

The **−r** flag causes the articles to be printed in reverse order. The **−f** flag prevents any followup articles from being printed. The **−h** flag causes articles to be printed in a less verbose format, and is intended for terminals running at 300 baud. The **−u** flag causes the *.newsrc* file to be updated 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.

Readnews maintains a *.newsrc* file in the user's home directory that specifies all news articles already read. It is updated at the end of each reading session in which the **−x** or **−l** options weren't specified. If the environment variable **NEWSRC** is present, it should be the path name of a file to be used in place of *.newsrc*.

If the user wishes, an options line may be placed in the *.newsrc* file. This line starts with the word **options** (left justified) followed by the list of standard options just as they would be typed on the command line. Such a list may include: the **−n** flag along with a newsgroup list; a favorite interface; 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. Where conflicts exist, option on the command line take precedence, followed by the *.newsrc* **options** line, and lastly the **NEWSOPTS** parameter.

You can use the **−s** flag to print the newsgroup subscription list.

When the user uses the reply command of the default or *binmail*(1) interfaces, the environment parameter **MAILER** will be used to determine which mailer to use. The default is *mail*(1).

If the user so desires, he may specify a specific paging progam for articles. The environment parameter **PAGER** should be set to the paging program. The name of the article is referenced with a ''%'', as in the **−c** option. If no ''%'' is present, the article will be piped to the program. Paging may be disabled by setting **PAGER** to a null value. By default, the pager is *cat*(1).

COMMANDS

This section lists the commands you can type to the default and *binmail* interface prompts. The default interface will suggest some common commands in brackets. Just hitting 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.

Command	Meaning
−	Go back to last article. This is a toggle, typing it twice returns you to the original article.
#	Report the name and size of the newsgroup.
!	Shell escape.
<*message ID***>**	Look for a particular article. (See *Standard for Interchange of Usenet Messages* for a description of message ID's).
b	Back. Back up one article.
c	Cancel the article. Only the author or the super user can do this.
d	Read a digest. Breaks up a digest into separate articles and permits you to 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. An explicit *number* rotation (usually 13) may be given to force a particular shift.
e	Erase. Forget that this article was read.
f [*title*]	Submit a follow up article. Normally you should leave off the title, since the system will generate one for you. You will be placed in your **EDITOR** to 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.
h	Header. Print a more verbose header.
H	Print a very verbose header, containing all known information about the article.
K	Kill. Mark all remaining articles in this newsgroup as read and skip to the next newsgroup.
n	No. Goes on to next article without printing current one. In the *binmail* interface, this means ''go on to the next article'', which will have the same effect as **y** or just hitting return.
N [*newsgroup*]	Next Newsgroup. Go to the next newsgroup or named newsgroup.
p	Print. Reprint previous article.
P	Previous Newsgroup. Go back to previous newsgroup.
q	Quit. The *.newsrc* file will be updated if **−l** or **−x** were not on the command line.
r	Reply. Reply to article's author via *mail*. You are placed in your **EDITOR** (by default *vi*(1)) 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. You are placed in **MAILER** (*mail* by default) in reply to the author. Type the text of the reply and then control-D.

s [*file*] Save. The article is appended to the named file. The default is *Articles*. If the first character of the file name is ''|'', the rest of the file name 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 file name is ''/'', it is taken as a full path name of a file. If **NEWSBOX** (in the environment) is set to a full path name, and the file contains no ''/'', the file is saved in **NEWSBOX**. Otherwise, it is saved relative to **HOME**.

U Unsubscribe from this newsgroup. Also goes on to the next newsgroup.

v Print the current version of the news software.

w Same as **s**.

x Exit. Like quit except that *.newsrc* is not updated.

X *system*
Transmit article to the named system.

y Yes. Prints current article and goes on to next.

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.

The commands **c**, **f**, **fd**, **r**, **rd**, **e**, **h**, **H**, and **s** can be followed by **−**'s to refer to the previous article. Thus, when replying to an article using the default 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 default interface. The *.newsrc* file is updated at the end of the session.

readnews −c ''ed %'' −l
Invoke 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 **fa.** via *mailx* 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.

ENVIRONMENT VARIABLES

EDITOR
Editor invoked by **f** command. (Default is */usr/ucb/vi*.)

MAILER
Mailing program invoked by the **r** command. (Default is */bin/mail*.)

NAME Your full name used in header of articles posted by you. (Default is the comments field of your id in */etc/passwd*.)

NEWSBOX
File or directory where articles saved with the **s** command are stored. (Default is same as

HOME.)

NEWSOPTS
Options for *readnews*.

ORGANIZATION
Full name of this site used header of articles posted by you.

PAGER
Paging program invoked by articles with more than 16 lines. (Default is */usr/ucb/more*.)

SHELL
The shell invoked by the **!** command. (Default is */bin/sh*.)

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 default interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

binmail(1), checknews(1N), inews(1N), mail(1), mailx(1N), news(5N), newsrc(5N), postnews(1N), vnews(1N), getdate(3N), expire(8N), recnews(8N), sendnews(8N), uurec(8N)
(many of these not in printed manuals, but available online)
How to Read the Network News by Mark Horton (USD:9)
Standard for Interchange of Usenet Messages by Mark Horton.

AUTHORS

Matt Glickman
Mark Horton
Stephen Daniel
Tom R. Truscott

NAME

refer − find and insert literature references in documents

SYNOPSIS

refer [**−a**] [**−b**] [**−c**] [**−e**] [**−f***n*] [**−k***x*] [**−l***m,n*] [**−n**] [**−p** bib] [**−s***keys*] [**−B***l.m*] [**−P**] [**−S**] [file ...]

DESCRIPTION

Refer is a preprocessor for *nroff* or *troff*(1) that finds and formats references for footnotes or endnotes. It is also the base for a series of programs designed to index, search, sort, and print stand-alone bibliographies, or other data entered in the appropriate form.

Given an incomplete citation with sufficiently precise keywords, *refer* will search a bibliographic database for references containing these keywords anywhere in the title, author, journal, etc. The input file (or standard input) is copied to standard output, except for lines between .[and .] delimiters, which are assumed to contain keywords, and are replaced by information from the bibliographic database. The user may also search different databases, override particular fields, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. By default references are flagged by footnote numbers.

The following options are available:

−a*n* Reverse the first *n* author names (Jones, J. A. instead of J. A. Jones). If *n* is omitted all author names are reversed.

−b Bare mode: do not put any flags in text (neither numbers nor labels).

−c*keys*
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *keys*.

−e Instead of leaving the references where encountered, accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to same source.

−f*n* Set the footnote number to *n* instead of the default of 1 (one). With labels rather than numbers, this flag is a no-op.

−k*x* Instead of numbering references, use labels as specified in a reference data line beginning *%x;* by default *x* is **L.**

−l*m***,***n* Instead of numbering references, use labels made 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 *m* or *n* is omitted the entire name or date respectively is used.

−n Do not search the default file /usr/dict/papers/Ind. If there is a REFER environment variable, the specified file will be searched instead of the default file; in this case the **−n** flag has no effect.

−p *bib*
Take the next argument *bib* as a file of references to be searched. The default file is searched last.

−s*keys* Sort references by fields whose key-letters are in the *keys* string; permute reference numbers in text accordingly. Implies **−e**. The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with **+** taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort, for example, on all authors and then title, use **-sA+T**.

−B*l.m* Bibliography mode. Take a file composed of records separated by blank lines, and turn them into *troff* input. Label *l* will be turned into the macro *.m* with *l* defaulting to **%X** and *.m* defaulting to **.AP** (annotation paragraph).

−P Place punctuation marks .,:;?! after the reference signal, rather than before. (Periods and commas used to be done with strings.)

−S Produce references in the Natural or Social Science format.

To use your own references, put them in the format described below. They can be searched more rapidly by running *indxbib*(1) on them before using *refer;* failure to index results in a linear search. When *refer* is used with the *eqn, neqn* or *tbl* preprocessors *refer* should be first, to minimize the volume of data passed through pipes.

The *refer* preprocessor and associated programs expect input from a file of references composed of records separated by blank lines. A record is a set of lines (fields), each containing one kind of information. Fields start on a line beginning with a ''%'', followed by a key-letter, then a blank, and finally the contents of the field, and continue until the next line starting with ''%''. The output ordering and formatting of fields is controlled by the macros specified for *nroff/troff* (for footnotes and endnotes) or *roffbib* (for stand-alone bibliographies). For a list of the most common key-letters and their corresponding fields, see *addbib*(1). An example of a *refer* entry is given below.

EXAMPLE

```
%A	M. E. Lesk
%T	Some Applications of Inverted Indexes on the UNIX System
%B	UNIX Programmer's Manual
%V	2b
%I	Bell Laboratories
%C	Murray Hill, NJ
%D	1978
```

FILES

/usr/dict/papers	directory of default publication lists
/usr/lib/refer	directory of companion programs

SEE ALSO

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

AUTHOR

Mike Lesk

BUGS

Blank spaces at the end of lines in bibliography fields will cause the records to sort and reverse incorrectly. Sorting large numbers of references causes a core dump.

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*] [**−8**] [**−L**] [**−l** username]
rhost [**−e***c*] [**−8**] [**−L**] [**−l** username]

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 is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid some security problems, the .rhosts file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment TERM variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. 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. The optional argument **−8** allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than ^S/^Q. The argument **−L** allows the rlogin session to be run in litout mode. A line of the form ''~.'' disconnects from the remote host, where ''~'' is the escape character. Similarly, the line ''~^Z'' (where ^Z, control-Z, is the suspend character) will suspend the rlogin session. Substitution of the delayed-suspend character (normally ^Y) for the suspend character suspends the send portion of the rlogin, but allows output from the remote system. A different escape character may be specified by the **−e** option. There is no space separating this option flag and the argument character.

SEE ALSO

rsh(1C)

FILES

/usr/hosts/* for *rhost* version of the command

BUGS

More of the environment should be propagated.

NAME

rm, rmdir − remove (unlink) files or directories

SYNOPSIS

rm [**−f**] [**−r**] [**−i**] [**−**] file ...

rmdir dir ...

DESCRIPTION

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.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

rm(1), unlink(2), rmdir(2)

NAME

rmail − handle remote mail received via uucp

SYNOPSIS

rmail user ...

DESCRIPTION

Rmail interprets incoming mail received via *uucp*(1C), collapsing ''From'' lines in the form generated by *binmail*(1) into a single line of the form ''return-path!sender'', and passing the processed mail on to *sendmail*(8).

Rmail is explicitly designed for use with *uucp* and *sendmail*.

SEE ALSO

binmail(1), uucp(1C), sendmail(8)

BUGS

Rmail should not reside in /bin.

NAME

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**] [**−T***term*] [**−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 **−r**N1 will number the references starting at one (1). The flag **−r**V2 will double space the bibliography, while **−r**V1 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 **−r**L6i argument, and the page offset can be set from the default of 0 to one inch by specifying **−r**O1i (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.

The remote username used is the same as your local username, unless you specify a different remote name with the **−l** option. 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.

If you omit *command,* then instead of executing a single command, you will be logged in on the remote host using *rlogin*(1C).

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

rsh otherhost cat remotefile >> localfile

appends the remote file *remotefile* to the localfile *localfile,* while

rsh otherhost cat remotefile ">>" otherremotefile

appends *remotefile* to *otherremotefile.*

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.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1C)

BUGS

If you are using *csh*(1) and put a *rsh*(1C) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired you should redirect the input of *rsh* to /dev/null using the **−n** option.

You cannot run an interactive command (like *rogue*(6) or *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**] [**−r**] [**−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 11 minutes are shown as being down.

Users idle an hour or more are not counted unless the **−a** flag is given.

Normally, the listing is sorted by host name. The **−l** **,** **−t** **,** and **−u** flags specify sorting by load average, uptime, and number of users, respectively. The **−r** flag reverses the sort order.

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,* but for all machines on the local network. If no report has been received from a machine for 5 minutes then *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a users hasn't typed to the system for a minute or more, then *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of *rwho* unless the **−a** flag is given.

FILES

/usr/spool/rwho/whod.*   information about other machines

SEE ALSO

ruptime(1C), rwhod(8C)

BUGS

This is unwieldy when the number of machines on the local net is large.

NAME

sccs − front end for the SCCS subsystem

SYNOPSIS

sccs [**−r**] [**−d***path*] [**−p***path*] command [flags] [args]

DESCRIPTION

Sccs is a front end to the SCCS programs that helps them mesh more cleanly with the rest of UNIX. It also includes the capability to run ‘‘set user id’’ to another user to provide additional protection.

Basically, *sccs* runs the *command* with the specified *flags* and *args.* Each argument is normally modified to be prepended with ‘‘SCCS/s.’’.

Flags to be interpreted by the *sccs* program must be before the *command* argument. Flags to be passed to the actual SCCS program must come after the *command* argument. These flags are specific to the command and are discussed in the documentation for that command.

Besides the usual SCCS commands, several ‘‘pseudo-commands’’ can be issued. These are:

edit — Equivalent to ‘‘get −e’’.

delget — Perform a delta on the named files and then get new versions. The new versions will have id keywords expanded, and will not be editable. The −m, −p, −r, −s, and −y flags will be passed to delta, and the −b, −c, −e, −i, −k, −l, −s, and −x flags will be passed to get.

deledit — Equivalent to ‘‘delget’’ except that the ‘‘get’’ phase includes the ‘‘−e’’ flag. This option is useful for making a ‘‘checkpoint’’ of your current editing phase. The same flags will be passed to delta as described above, and all the flags listed for ‘‘get’’ above except −e and −k are passed to ‘‘edit’’.

create — Creates an SCCS file, taking the initial contents from the file of the same name. Any flags to ‘‘admin’’ are accepted. If the creation is successful, the files are renamed with a comma on the front. These should be removed when you are convinced that the SCCS files have been created successfully.

fix — Must be followed by a **−r** flag. This command essentially removes the named delta, but leaves you with a copy of the delta with the changes that were in it. It is useful for fixing small compiler bugs, etc. Since it doesn't leave audit trails, it should be used carefully.

clean — This routine removes everything from the current directory that can be recreated from SCCS files. It will not remove any files being edited. If the **−b** flag is given, branches are ignored in the determination of whether they are being edited; this is dangerous if you are keeping the branches in the same directory.

unedit — This is the opposite of an ‘‘edit’’ or a ‘‘get −e’’. It should be used with extreme caution, since any changes you made since the get will be irretrievably lost.

info — Gives a listing of all files being edited. If the **−b** flag is given, branches (i.e., SID's with two or fewer components) are ignored. If the **−u** flag is given (with an optional argument) then only files being edited by you (or the named user) are listed.

check — Like ‘‘info’’ except that nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. The intent is to have this included in an ‘‘install’’ entry in a makefile to insure that everything is included into the SCCS file before a version is installed.

tell — Gives a newline-separated list of the files being edited on the standard output. Takes the **−b** and **−u** flags like ‘‘info’’ and ‘‘check’’.

diffs — Gives a ‘‘diff’’ listing between the current version of the program(s) you have out for editing and the versions in SCCS format. The **−r, −c, −i, −x,** and **−t** flags are passed to *get*; the **−l, −s, −e, −f, −h,** and **−b** options are passed to *diff.* The **−C** flag is passed to *diff* as **−c.**

print This command prints out verbose information about the named files.

The **−r** flag runs *sccs* as the real user rather than as whatever effective user *sccs* is ''set user id'' to. The **−d** flag gives a root directory for the SCCS files. The default is the current directory. The **−p** flag defines the pathname of the directory in which the SCCS files will be found; ''SCCS'' is the default. The **−p** flag differs from the **−d** flag in that the **−d** argument is prepended to the entire pathname and the **−p** argument is inserted before the final component of the pathname. For example, ''sccs −d/x −py get a/b'' will convert to ''get /x/a/y/s.b''. The intent here is to create aliases such as ''alias syssccs sccs -d/usr/src'' which will be used as ''syssccs get cmd/who.c''. Also, if the environment variable PROJECT is set, its value is used to determine the **−d flag.** If it begins with a slash, it is taken directly; otherwise, the home directory of a user of that name is examined for a subdirectory ''src'' or ''source''. If such a directory is found, it is used.

Certain commands (such as *admin*) cannot be run ''set user id'' by all users, since this would allow anyone to change the authorizations. These commands are always run as the real user.

EXAMPLES

To get a file for editing, edit it, and produce a new delta:

```
sccs get −e file.c
ex file.c
sccs delta file.c
```

To get a file from another directory:

```
sccs −p/usr/src/sccs/s. get cc.c
```

or

```
sccs get /usr/src/sccs/s.cc.c
```

To make a delta of a large number of files in the current directory:

```
sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
sccs info −b
```

To delta everything being edited by you:

```
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) $@
```

SEE ALSO

admin(SCCS), chghist(SCCS), comb(SCCS), delta(SCCS), get(SCCS), help(SCCS), prt(SCCS), rmdel(SCCS), sccsdiff(SCCS), what(SCCS)
Eric Allman, *An Introduction to the Source Code Control System*

BUGS

It should be able to take directory arguments on pseudo-commands like the SCCS commands do.

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.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

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. The **−f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **−e** option and no **−f**'s, the flag **−e** may be omitted. The **−n** option suppresses the default output.

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a 'D' command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **−n**) and deletes the pattern space.

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*. If *label* is empty, branch to the end of the script.

(2) c\
text

Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2) D Delete the initial segment of the pattern space through the first 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)

NAME

sendbug − mail a system bug report to 4bsd-bugs

SYNOPSIS

sendbug [address]

DESCRIPTION

Bug reports sent to '4bsd-bugs@Berkeley.EDU' are intercepted by a program which expects bug reports to conform to a standard format. *Sendbug* is a shell script to help the user compose and mail bug reports in the correct format. *Sendbug* works by invoking the editor specified by the environment variable *EDITOR* on a temporary copy of the bug report format outline. The user must fill in the appropriate fields and exit the editor. The default editor is *vi*(1). *Sendbug* then mails the completed report to '4bsd-bugs@Berkeley.EDU' or the address specified on the command line.

FILES

/usr/ucb/bugformat contains the bug report outline

SEE ALSO

vi(1), environ(7), sendmail(8)

NAME

sh, for, case, if, while, **:**, **.**, break, continue, cd, eval, exec, exit, export, login, read, readonly, set, shift, times, trap, umask, wait − command language

SYNOPSIS

sh [**−ceiknrstuvx**] [arg] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See **invocation** for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non blank *words* separated by blanks (a blank is a **tab** or a **space**). The first word specifies the name of the command to be executed. Except as specified below the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *execve*(2)). The *value* of a simple-command is its exit status if it terminates normally or 200+*status* if it terminates abnormally (see *sigvec*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|.** The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more *pipelines* separated by **;**, **&**, **&&** or || and optionally terminated by **;** or **&**. **;** and **&** have equal precedence which is lower than that of **&&** and ||, **&&** and || also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol **&&** (||) causes the *list* following to be executed only if the preceding *pipeline* returns a zero (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.

for *name* [**in** *word* ...] **do** *list* **done**
: Each time a **for** command is executed *name* is set to the next word in the **for** 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* **;;**] ... **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 in 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**. **IFS** is ignored if *sh* is running as root or if the effective user id differs from the real user id.

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 ****. **\\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 **\\ ´ "** and **$**.

"$*" is equivalent to **"$1 $2 ..."** whereas
"$@" is equivalent to **"$1" "$2" ... .**

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

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

< *word* Use file *word* as standard input (file descriptor 0).

> *word* Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise it is truncated to zero length.

>> *word* Use file *word* as standard output. If the file exists, output is appended (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 **\\ $ ´** 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*(7). 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 the 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 (i.e., 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.

For non-interactive shells, everything following the **#** is treated as a comment, i.e. the rest of the line is ignored. For interactive shells, the **#** has no special effect.

: 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 ...'.

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 interruptible). 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

csh(1), test(1), execve(2), environ(7)

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), alarm(3C), 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 '.', i.e.

´so /usr/lib/tmac.s

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 blank must precede and no blanks follow the file name.

NAME

sort − sort or merge files

SYNOPSIS

sort [**−mubdfinrt***x*] [**+***pos1* [**−***pos2*]] ... [**−o** name] [**−T** directory] [name] ...

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 files 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 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*.

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.

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 The next argument 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 The next 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 [**−s**KEYS] 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, **−s**ATD will sort by author, title, and date, while **−s**A+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)

AUTHORS

Greg Shenaut, Bill Tuthill

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell, spellin, spellout − find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−x**] [**−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, tbl* and *eqn*(1) constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the **−b** option, British spelling is checked. Besides preferring *centre, colour, speciality, travelled,* etc., this option insists upon *-ise* in words like *standardise,* Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with '=' for each word.

The spelling list is based on many sources. While it is more haphazard than an ordinary dictionary, it is also more effective with proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

The auxiliary files used for the spelling list, stop list, and history file may be specified by arguments following the **−d**, **−s**, and **−h** options. The default files are indicated below. Copies of all output may be accumulated in the history file. The stop list filters out misspellings (e.g. thier=thy−y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell.* Both expect a set of words, one per line, from the standard input. *Spellin* combines the words from the standard input and the preexisting *list* file and places a new list on the standard output. If no *list* file is specified, the new list is created from scratch. *Spellout* looks up each word from the standard input and prints on the standard output those that are missing from (or present on, with option **−d**) the hashed *list* file. For example, to verify that *hookey* is not on the default spelling list, add it to your own private list, and then use it with *spell*,

```
echo hookey | spellout /usr/dict/hlista
echo hookey | spellin /usr/dict/hlista > myhlist
spell −d myhlist huckfinn
```

FILES

/usr/dict/hlist[ab] hashed spelling lists, American & British, default for **−d**
/usr/dict/hstop hashed stop list, default for **−s**
/dev/null history file, default for **−h**
/tmp/spell.$$∗ temporary files
/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 [option] ...

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).

The following options are recognized, each as a separate argument.

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

−k The constant *k* used in the boundary value computation

$$y_0'' = k y_1'', \quad y_n'' = k y_{n-1}''$$

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, i.e. match derivatives at ends. First and last input values should normally agree.

−x Next 1 (or 2) arguments are lower (and upper) *x* limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1G), plot(1G)

DIAGNOSTICS

When data is not strictly monotone 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), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if **−** is given in its stead, then the standard input file is used.

NAME

strings − find the printable strings in a 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. Unless the − flag is given, *strings* only looks in the initialized data space of object files. If the **−o** flag is given, then each string is preceded by its offset in the file (in octal). If the −*number* flag is given then number is used as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

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

struct − structure Fortran programs

SYNOPSIS

struct [option] ... file

DESCRIPTION

Struct translates the Fortran program specified by *file* (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original Fortran. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (.e.g. ".GT." into ">"). The output is appropriately indented.

The following options may occur in any order.

−s Input is accepted in standard format, i.e. comments are specified by a c, C, or * in column 1, and continuation lines are specified by a nonzero, nonblank character in column 6. Normally input is in the form accepted by *f77*(1)

−i Do not turn computed goto statements into switches. (Ratfor does not turn switches back into computed goto statements.)

−a Turn sequences of else ifs into a non-Ratfor switch of the form

```
switch
        {   case pred1: code
            case pred2: code
            case pred3: code
            default: code
        }
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

−b Generate goto's instead of multilevel break statements.

−n Generate goto's instead of multilevel next statements.

−t*n* Make the nonzero integer *n* the lowest valued label in the output program (default 10).

−c*n* Increment successive labels in the output program by the nonzero integer *n* (default 1).

−e*n* If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit a small code segments to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop. 'Small' is close to, but not equal to, the number of statements in the code segment. Values of n under 10 are suggested.

FILES

/tmp/struct*
/usr/lib/struct/*

SEE ALSO

f77(1)

BUGS

Struct knows Fortran 66 syntax, but not full Fortran 77.
If an input Fortran program contains identifiers which are reserved words in Ratfor, the structured version of the program will not be a valid Ratfor program.
The labels generated cannot go above 32767.
If you get a goto without a target, try **−e .**

NAME

stty − set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, placing its output on the diagnostic output. With no argument, it reports the speed of the terminal and the settings of the options which are different from their defaults. Use of one of the following options modifies the output as described:

all All normally used option settings are reported.

everything Everything *stty* knows about is printed.

speed The terminal speed alone is printed on the standard output.

size The terminal (window) sizes are printed on the standard output, first rows and then columns.

The option strings 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 # and @

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 '#', but often reset to ^H.)
kill *c* set kill character to *c* (default '@', but often reset to ^U.)
intr *c* set interrupt character to *c* (default DEL or ^? (delete), but often reset to ^C.)
quit *c* set quit character to *c* (default control \.)
start *c* set start character to *c* (default control Q.)
stop *c* set stop character to *c* (default control S.)
eof *c* set end of file character to *c* (default control D.)
brk *c* set break character to *c* (default undefined.) This character is an additional character causing wakeup.

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).

rows *n* The terminal size is recorded as having *n* rows.

columns *n* The terminal size is recorded as having *n* columns.

cols *n* is an alias for *columns*.

A teletype driver which supports the job control processing of *csh*(1) and more functionality than the basic driver is fully described in *tty*(4). The following options apply only to it.

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 (control D).

−ctlecho Control characters echo as themselves; in cooked mode EOT (control-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 control 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.

pass8	Passes all 8 bits through on input, in any mode.
−pass8	Strips the 0200 bit on input except in raw mode.
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 control Z).
dsusp *c*	set delayed suspend process character to *c* (default control Y).
rprnt *c*	set reprint line character to *c* (default control R).
flush *c*	set flush output character to *c* (default control O).
werase *c*	set word erase character to *c* (default control W).
lnext *c*	set literal next character to *c* (default control V).

SEE ALSO

ioctl(2), tabs(1), tset(1), tty(4)

NAME

style − analyze surface characteristics of a document

SYNOPSIS

style [**−ml**] [**−mm**] [**−a**] [**−e**] [**−l** num] [**−r** num] [**−p**] [**−P**] file ...

DESCRIPTION

Style analyzes the surface characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because *style* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **−ms** may be overridden with the flag **−mm.** The flag **−ml**, which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The other options are used to locate sentences with certain characteristics.

−a print all sentences with their length and readability index.

−e print all sentences that begin with an expletive.

−p print all sentences that contain a passive verb.

−l*num* print all sentences longer than *num*.

−r*num* print all sentences whose readability index is greater than *num*.

−P print parts of speech of the words in the document.

SEE ALSO

deroff(1), diction(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

NAME

su − substitute user id temporarily

SYNOPSIS

su [−f] [−] [userid]

DESCRIPTION

Su demands the password of the specified *userid,* and if it is given, changes to that *userid* and invokes the Shell *sh*(1) or *csh*(1) without changing the current directory. The user environment is unchanged except for HOME and SHELL, which are taken from the password file for the user being substituted (see *environ*(7)). The new user ID stays in force until the Shell exits.

If no *userid* is specified, ''root'' is assumed. Only users in the ''wheel'' group (group 0) can *su* to ''root'', even with the root password. To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

The **−f** option prevents *csh*(1) from executing the .cshrc file; thus making *su* start up faster.

The **−** option simulates a full login.

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

symorder − rearrange name list

SYNOPSIS

symorder orderlist symbolfile

DESCRIPTION

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

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

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

SEE ALSO

nlist(3)

NAME

sysline − display system status on status line of a terminal

SYNOPSIS

sysline [**−bcdewhDilmpqrsj**] [**−H remote**] [**+N**]

DESCRIPTION

Sysline runs in the background and periodically displays system status information on the status line of the terminal. Not all terminals contain a status line. Those that do include the h19, concept 108, Ann Arbor Ambassador, vt100, Televideo 925/950 and Freedom 100. If no flags are given, *sysline* displays the time of day, the current load average, the change in load average in the last 5 minutes, the number of users (followed by a 'u'), the number of runnable process (followed by a 'r')[VAX only], the number of suspended processes (followed by a 's')[VAX only], and the users who have logged on and off since the last status report. Finally, if new mail has arrived, a summary of it is printed. If there is unread mail in your mailbox, an asterisk will appear after the display of the number of users. The display is normally in reverse video (if your terminal supports this in the status line) and is right justified to reduce distraction. Every fifth display is done in normal video to give the screen a chance to rest.

If you have a file named .who in your home directory, then the contents of that file is printed first. One common use of this feature is to alias chdir, pushd, and popd to place the current directory stack in ~/.who after it changes the new directory.

The following flags may be given on the command line.

−b Beep once every half hour and twice every hour, just like those obnoxious watches you keep hearing.

−c Clear the status line for 5 seconds before each redisplay.

−d Debug mode -- print status line data in human readable format.

−D Print out the current day/date before the time.

−e Print out only the information. Do not print out the control commands necessary to put the information on the bottom line. This option is useful for putting the output of *sysline* onto the mode line of an emacs window.

−w Window mode -- print the status on the current line of the terminal, suitable for use inside a one line window.

−H remote Print the load average on the remote host *remote* [VAX only]. If the host is down, or is not sending out *rwhod* packets, then the down time is printed instead. If the prefix "ucb" is present, then it is removed.

−h Print out the host machine's name after the time [VAX only].

−l Don't print the names of people who log in and out.

−m Don't check for mail.

−p Don't report the number of process which are runnable and suspended.

−r Don't display in reverse video.

+N Update the status line every N seconds. The default is 60 seconds.

−q Don't print out diagnostic messages if something goes wrong when starting up.

−i Print out the process id of the *sysline* process onto standard output upon startup. With this information you can send the alarm signal to the *sysline* process to cause it to update immediately. *sysline* writes to the standard error, so you can redirect the standard output into a file to catch the process id.

−s Print "short" form of line by left-justifying *iff* escapes are not allowed in the status line. Some terminals (the Televideos and Freedom 100 for example) do not allow cursor movement (or other "intelligent" operations) in the status line. For these terminals, *sysline* normally uses blanks to cause right-justification. This flag will disable the adding of the blanks.

−j Force the sysline output to be left justified even on terminals capable of cursor movement on the status line.

If you have a file .syslinelock in your home directory, then *sysline* will not update its statistics and write on your screen, it will just go to sleep for a minute. This is useful if you want to momentarily disable *sysline*. Note that it may take a few seconds from the time the lock file is created until you are guaranteed that *sysline* will not write on the screen.

FILES

/etc/utmp	names of people who are logged in
/dev/kmem	contains process table [VAX only]
/usr/spool/rwho/whod.*	who/uptime information for remote hosts [VAX only]
${HOME}/.who	information to print on bottom line
${HOME}/.syslinelock	when it exists, sysline will not print

AUTHORS

John Foderaro
Tom Ferrin converted it to use termcap.
Mark Horton added terminfo capability.

BUGS

If you interrupt the display then you may find your cursor missing or stuck on the status line. The best thing to do is reset the terminal.
If there is too much for one line, the excess is thrown away.

NAME

systat − display system statistics on a crt

SYNOPSIS

systat [−*display*] [refresh-interval]

DESCRIPTION

Systat displays various system statistics in a screen oriented fashion using the curses screen display library, *curses*(3X).

While *systat* is running the screen is usually divided into two windows (an exception is the vmstat display which uses the entire screen). The upper window depicts the current system load average. The information displayed in the lower window may vary, depending on user commands. The last line on the screen is reserved for user input and error messages.

By default *systat* displays the processes getting the largest percentage of the processor in the lower window. Other displays show swap space usage, disk i/o statistics (a la *iostat*(1)), virtual memory statistics (a la *vmstat*(1)), network ''mbuf'' utilization, and network connections (a la *netstat*(1)).

Input is interpreted at two different levels. A ''global'' command interpreter processes all keyboard input. If this command interpreter fails to recognize a command, the input line is passed to a per-display command interpreter. This allows each display to have certain display-specific commands.

Certain characters cause immediate action by *systat*. These are

^L Refresh the screen.

^G Print the name of the current ''display'' being shown in the lower window and the refresh interval.

^Z Stop *systat*.

: Move the cursor to the command line and interpret the input line typed as a command. While entering a command the current character erase, word erase, and line kill characters may be used.

The following commands are interpreted by the ''global'' command interpreter.

help
: Print the names of the available displays on the command line.

load
: Print the load average over the past 1, 5, and 15 minutes on the command line.

stop
: Stop refreshing the screen.

[start] [number]
: Start (continue) refreshing the screen. If a second, numeric, argument is provided it is interpreted as a refresh interval (in seconds). Supplying only a number will set the refresh interval to this value.

quit
: Exit *systat*. (This may be abbreviated to *q*.)

The available displays are:

pigs
: Display, in the lower window, those processes resident in main memory and getting the largest portion of the processor (the default display). When less than 100% of the processor is scheduled to user processes, the remaining time is accounted to the ''idle'' process.

iostat
: Display, in the lower window, statistics about processor use and disk throughput. Statistics on processor use appear as bar graphs of the amount of time executing in user mode (''user''), in

user mode running low priority processes (''nice''), in system mode (''system''), and idle (''idle''). Statistics on disk throughput show, for each drive, kilobytes of data transferred, number of disk transactions performed, and average seek time (in milliseconds). This information may be displayed as bar graphs or as rows of numbers which scroll downward. Bar graphs are shown by default; commands specific to this display are discussed below.

swap

Display, in the lower window, swap space in use on each swap device configured. Two sets of bar graphs are shown. The upper graph displays swap space allocated to pure text segments (code), the lower graph displays space allocated to stack and data segments. Allocated space is sorted by its size into buckets of size dmmin, dmmin*2, dmmin*4, up to dmmax (to reflect allocation policies imposed by the system). The disk segment size, in sectors, is displayed along the left hand side of the text, and data and stack graphs. Space allocated to the user structure and page tables is not currently accounted for.

mbufs

Display, in the lower window, the number of mbufs allocated for particular uses, i.e. data, socket structures, etc.

vmstat

Take over the entire display and show a (rather crowded) compendium of statistics related to virtual memory usage, process scheduling, device interrupts, system name translation cacheing, disk i/o, etc.

The upper left quadrant of the screen shows the number of users logged in and the load average over the last one, five, and fifteen minute intervals. Below this line are statistics on memory utilization. The first row of the table reports memory usage only among active processes, that is processes that have run in the previous twenty seconds. The second row reports on memory usage of all processes. The first column reports on the number of physical pages claimed by processes. The second column reports the number of physical pages that are devoted to read only text pages. The third and fourth columns report the same two figures for virtual pages, that is the number of pages that would be needed if all processes had all of their pages. Finally the last column shows the number of physical pages on the free list.

Below the memory display is the disk usage display. It reports the number of seeks, transfers, and number of kilobyte blocks transferred per second averaged over the refresh period of the display (by default, five seconds). For some disks it also reports the average milliseconds per seek. Note that the system only keeps statistics on at most four disks.

Below the disk display is a list of the average number of processes (over the last refresh interval) that are runnable ('r'), in page wait ('p'), in disk wait other than paging ('d'), sleeping ('s'), and swapped out but desiring to run ('w'). Below the queue length listing is a numerical listing and a bar graph showing the amount of system (shown as '='), user (shown as '>'), nice (shown as '-'), and idle time (shown as ' ').

At the bottom left are statistics on name translations. It lists the number of names translated in the previous interval, the number and percentage of the translations that were handled by the system wide name translation cache, and the number and percentage of the translations that were handled by the per process name translation cache.

Under the date in the upper right hand quadrant are statistics on paging and swapping activity. The first two columns report the average number of pages brought in and out per second over the last refresh interval due to page faults and the paging daemon. The third and fourth columns report the average number of pages brought in and out per second over the last refresh interval due to swap requests initiated by the scheduler. The first row of the display shows the average number of disk transfers per second over the last refresh interval; the second row of the display shows the average number of pages transferred per second over the last refresh interval.

Below the paging statistics is a line listing the average number of total reclaims ('Rec'), intransit blocking page faults ('It'), swap text pages found in free list ('F/S'), file system text pages found in free list ('F/F'), reclaims from free list ('RFL'), pages freed by the clock daemon ('Fre'), and sequential process pages freed ('SFr') per second over the refresh interval.

Below this line are statistics on the average number of zero filled pages ('zf') and demand filled text pages ('xf') per second over the refresh period. The first row indicates the number of requests that were resolved, the second row shows the number that were set up, and the last row shows the percentage of setup requests were actually used. Note that this percentage is usually less than 100%, however it may exceed 100% if a large number of requests are actually used long after they were set up during a period when no new pages are being set up. Thus this figure is most interesting when observed over a long time period, such as from boot time (see below on getting such a display).

Below the page fill statistics is a column that lists the average number of context switches ('Csw'), traps ('Trp'), system calls ('Sys'), interrupts ('Int'), characters output to DZ ports using pseudo-DMA ('Pdm'), page faults ('Flt'), pages scanned by the page daemon ('Scn'), and revolutions of the page daemon's hand ('Rev') per second over the refresh interval.

Running down the right hand side of the display is a breakdown of the interrupts being handled by the system. At the top of the list is the total interrupts per second over the time interval. The rest of the column breaks down the total on a device by device basis. Only devices that have interrupted at least once since boot time are shown.

netstat
: Display, in the lower window, network connections. By default, network servers awaiting requests are not displayed. Each address is displayed in the format ''host.port'', with each shown symbolically, when possible. It is possible to have addresses displayed numerically, limit the display to a set of ports, hosts, and/or protocols; see the list of commands below.

Commands to switch between displays may be abbreviated to the minimum unambiguous prefix; for example, ''io'' for ''iostat''. Certain information may be discarded when the screen size is insufficient for display. For example, on a machine with 10 drives the *iostat* bar graph displays only 3 drives on a 24 line terminal. When a bar graph would overflow the allotted screen space it is truncated and the actual value is printed ''over top'' of the bar.

The following commands are specific to the *iostat* display; the minimum unambiguous prefix may be supplied.

numbers
: Show the disk i/o statistics in numeric form. Values are displayed in numeric columns which scroll downward.

bars — Show the disk i/o statistics in bar graph form (default).

msps — Toggle the display of average seek time (the default is to not display seek times).

The following commands are specific to the *vmstat* display; the minimum unambiguous prefix may be supplied.

boot — Display cumulative statistics since the system was booted.

run — Display statistics as a running total from the point this command is given.

time — Display statistics averaged over the refresh interval (the default).

zero — Reset running statistics to zero.

The following commands are common to each display which shows information about disk drives. These commands are used to select a set of drives to report on, should your system have more drives configured than can normally be displayed on the screen.

ignore [drives]
: Do not display information about the drives indicated. Multiple drives may be specified, separated by spaces.

display [drives]
: Display information about the drives indicated. Multiple drives may be specified, separated by spaces.

The following command is specific to the *netstat* display; the minimum unambiguous prefix may be supplied.

all
: Toggle the displaying of server processes awaiting requests (this is the equivalent of the **−a** flag to *netstat*(1)).

numbers
: Display network addresses numerically.

names
: Display network addresses symbolically.

The remaining commands are common to displays which report network connections (currently only the *netstat* display). These commands may be used to select a specific set of connections for *systat* to report on.

protocol
: Display only network connections using the indicated protocol (currently either ''tcp'' or ''udp'').

ignore [items]
: Do not display information about connections associated with the specified hosts or ports. Hosts and ports may be specified by name (''ucbmonet'', ''ftp''), or numerically. Host addresses use the Internet dot notation (''128.32.0.9''). Multiple items may be specified with a single command by separating them with spaces.

display [items]
: Display information about the connections associated with the specified hosts or ports. As for *ignore*, *items* may be names or numbers.

show [ports|hosts]
: Show, on the command line, the currently selected protocols, hosts, and ports. Hosts and ports which are being ignored are prefixed with a '!'. If *ports* or *hosts* is supplied as an argument to *show*, then only the requested information will be displayed.

reset
: Reset the port, host, and protocol matching mechanisms to the default (any protocol, port, or host).

FILES

/vmunix	for the namelist
/dev/kmem	for information in main memory
/dev/drum	for information about swapped out processes
/etc/hosts	for host names
/etc/networks	for network names
/etc/services	for port names

AUTHOR

The unknown hacker. The *pigs* display is derived from a program of the same name written by Bill Reeves.

BUGS

Takes 2-10 percent of the cpu. Certain displays presume a 24 line by 80 character terminal. The swap space display should account for space allocated to the user structure and page tables. The *vmstat* display looks out of place because it is (it was added in as a separate display rather than create a new program).

The whole thing is pretty hokey and was included in the distribution under serious duress.

NAME

tabs − set terminal tabs

SYNOPSIS

tabs [**−n**] [terminal]

DESCRIPTION

Tabs sets the tabs on a variety of terminals. Various terminal names given in *term*(7) are recognized; the default is, however, suitable for most 300 baud terminals. If the **−n** flag is present then the left margin is not indented as is normal.

SEE ALSO

stty(1), term(7)

BUGS

It's much better to use *tset*(1).

NAME

tail − deliver 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.

Copying begins at distance +*number* from the beginning, or −*number* from the end of the input. *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.

Specifying **r** causes tail to print lines from the end of the file in reverse order. The default for **r** is to print the entire file this way. Specifying **f** causes *tail* to not 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 you 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 *host@user* 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, it 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

talk your_name@your_machine

It doesn't matter from which machine the recipient replies, as long as his 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 will cause the screen to be reprinted, 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* command. At the outset talking is allowed. Certain commands, in particular *nroff* 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)

BUGS

The version of *talk*(1) released with 4.3BSD uses a protocol that is incompatible with the protocol used in the version released with 4.2BSD.

NAME

tar − tape archiver

SYNOPSIS

tar [key] [name ...]

DESCRIPTION

Tar saves and restores multiple files on a single file (usually a magnetic tape, but it can be any file). *Tar*'s actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to *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.

The function portion of the key is specified by one of the following letters:

r The named files are written on the end of the tape. The **c** function implies this.

x The named files are extracted from the tape. 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.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape.

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies **r**.

The following characters may be used in addition to the letter which selects the function desired.

o On output, tar normally places information specifying owner and modes of directories in the archive. Former versions of tar, when encountering this information will give error message of the form

"<name>/: cannot create".

This modifier will suppress the directory information.

p This modifier says to restore files to their original modes, ignoring the present *umask*(2). Setuid and sticky information will also be restored to the super-user.

0, ..., 9 This modifier selects an alternate drive on which the tape is mounted. The default is drive 0 at 1600 bpi, which is normally /dev/rmt8.

v Normally *tar* does its work silently. The **v** (verbose) option makes *tar* print the name of each file it treats preceded by the function letter. With the **t** function, the verbose option gives more information about the tape entries than just their names.

w *Tar* prints the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is done. Any other input means don't do it.

f *Tar* uses the next argument as the name of the archive instead of /dev/rmt?. 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 move hierarchies with the command

cd fromdir; tar cf - . | (cd todir; tar xf -)

b *Tar* uses the next argument as the blocking factor for tape records. The default is 20 (the maximum). This option should only be used with raw magnetic tape archives (See **f** above). The block size is determined automatically when reading tapes (key letters 'x' and 't').

l tells *tar* to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m tells *tar* not to restore the modification times. The modification time will be the time of extraction.

h Force *tar* to follow symbolic links as if they were normal files or directories. Normally, *tar* does not follow symbolic links.

B Forces input and output blocking to 20 blocks per record. This option was added so that *tar* can work across a communications channel where the blocking may not be maintained.

C If a file name is preceded by **−C**, then *tar* will perform a *chdir*(2) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from /usr/include and from /etc, one might use

```
tar c -C /usr include -C / etc
```

Previous restrictions dealing with *tar*'s inability to properly handle blocked archives have been lifted.

FILES

/dev/rmt?
/tmp/tar*

SEE ALSO

tar(5)

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The current limit on file name length is 100 characters.
There is no way selectively to follow symbolic links.
When extracting tapes created with the **r** or **u** options, directory modification times may not be set correctly.

NAME

tbl − format tables for nroff or troff

SYNOPSIS

tbl [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input files are copied to the standard output, except for lines between and are reformatted. Details are given in the *tbl*(1) reference manual.

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

If no arguments are given, *tbl* reads the standard input, so it may be used as a filter. When *tbl* is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

SEE ALSO

troff(1), eqn(1)
M. E. Lesk, *TBL.*

NAME

tc − photoypesetter simulator

SYNOPSIS

tc [**−t**] [**−s**N] [**−p**L] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Graphic Systems phototypesetter (cat). The standard output of *tc* is intended for a Tektronix 4015 (a 4014 terminal with ASCII and APL character sets). The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary. Typical usage:

troff −t file | tc

At the end of each page *tc* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will suppress the screen erase before the next page; **s**N will cause the next N pages to be skipped; and **!**line will send line to the shell.

The command line options are:

−t Don't wait between pages; for directing output into a file.

−sN Skip the first N pages.

−pL Set page length to L. L may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

′−l w′ Multiply the default aspect ratio, 1.5, of a displayed page by *l/w*.

SEE ALSO

troff(1), plot(1G)

BUGS

Font distinctions are lost.
tc's character set is limited to ASCII in just one size.
The aspect ratio option is unbelievable.

NAME

tcopy − copy a mag tape

SYNOPSIS

tcopy src [**dest**]

DESCRIPTION

Tcopy is designed to copy magnetic tapes. The only assumption made about the tape is that there are two tape marks at the end. *Tcopy* with only a source tape specified will print information about the sizes of records and tape files. If a destination is specified, then, a copy will be made of the source tape. The blocking on the destination tape will be identical to that used on the source tape. Copying a tape will yield the same output as if just printing the sizes.

SEE ALSO

mtio(4)

NAME

tee − pipe fitting

SYNOPSIS

tee [**−i**] [**−a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. Option **−i** ignores interrupts; option **−a** causes the output to be appended to the *files* rather than overwriting them.

NAME

telnet − user interface to the TELNET protocol

SYNOPSIS

telnet [host [port]]

DESCRIPTION

Telnet is used to communicate 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 an input mode. The input mode entered will be either ''character at a time'' or ''line by line'' depending on what the remote system supports.

In ''character at a time'' mode, most text typed is immediately sent to the remote host for processing.

In ''line by line'' mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The ''local echo character'' (initially ''^E'') may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's *quit*, *intr*, and *flush* characters are trapped locally, and sent as **TELNET** protocol sequences to the remote side. There are options (see **toggle** *autoflush* and **toggle** *autosynch* below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the **TELNET** sequence) and flush previous terminal input (in the case of *quit* and *intr*).

While connected to a remote host, *telnet* command mode may be entered by typing the *telnet* ''escape character'' (initially ''^]''). When in command mode, the normal terminal editing conventions are available.

COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

open *host* [*port*]

Open a connection to the named host. If 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'' (see *inet*(3N)).

close

Close a **TELNET** session and return to command mode.

quit

Close any open **TELNET** session and exit *telnet*. An end of file (in command mode) will also close a session and exit.

z

Suspend *telnet*. This command only works when the user is using the *csh*(1).

mode *type*

Type is either *line* (for ''line by line'' mode) or *character* (for ''character at a time'' mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

status

Show the current status of *telnet*. This includes the peer one is connected to, as well as the current mode.

display [*argument...*]

Displays all, or some, of the **set** and **toggle** values (see below).

? [*command*]

Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information for just that command.

send *arguments*

Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

escape

Sends the current *telnet* escape character (initially ''^]'').

synch

Sends the **TELNET SYNCH** sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it doesn't work, a lower case ''r'' may be echoed on the terminal).

brk

Sends the **TELNET BRK** (Break) sequence, which may have significance to the remote system.

ip

Sends the **TELNET IP** (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao

Sends the **TELNET AO** (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

ayt

Sends the **TELNET AYT** (Are You There) sequence, to which the remote system may or may not choose to respond.

ec

Sends the **TELNET EC** (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el

Sends the **TELNET EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga

Sends the **TELNET GA** (Go Ahead) sequence, which likely has no significance to the remote system.

nop

Sends the **TELNET NOP** (No OPeration) sequence.

?

Prints out help information for the **send** command.

set *argument value*

Set any one of a number of *telnet* variables to a specific value. The special value ''off'' turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

echo

This is the value (initially ''^E'') which, when in ''line by line'' mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

escape

This is the *telnet* escape character (initially ``^['') which causes entry into *telnet* command mode (when connected to a remote system).

interrupt

If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *interrupt* character is typed, a **TELNET IP** sequence (see **send** *ip* above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

quit

If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed, a **TELNET BRK** sequence (see **send** *brk* above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

flushoutput

If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed, a **TELNET AO** sequence (see **send** *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

erase

If *telnet* is in *localchars* mode (see **toggle** *localchars* below), **and** if *telnet* is operating in ``character at a time'' mode, then when this character is typed, a **TELNET EC** sequence (see **send** *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

kill

If *telnet* is in *localchars* mode (see **toggle** *localchars* below), **and** if *telnet* is operating in ``character at a time'' mode, then when this character is typed, a **TELNET EL** sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

eof

If *telnet* is operating in ``line by line'' mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

toggle *arguments...*

Toggle (between TRUE and FALSE) various flags that control how *telnet* responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

localchars

If this is TRUE, then the *flush*, *interrupt*, *quit*, *erase*, and *kill* characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate **TELNET** control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in ``line by line'' mode, and FALSE in ``character at a time'' mode.

autoflush

If *autoflush* and *localchars* are both TRUE, then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into **TELNET** sequences; see **set** above for details), *telnet* refuses to display any data on the user's terminal until the remote system acknowledges (via a **TELNET** *Timing Mark* option) that it has processed those **TELNET** sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh", otherwise FALSE (see *stty(1)).*

autosynch

If *autosynch* and *localchars* are both TRUE, then when either the *intr* or *quit* characters is typed (see **set** above for descriptions of the *intr* and *quit* characters), the resulting **TELNET** sequence sent is followed by the **TELNET SYNCH** sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the **TELNET** sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod

Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

debug

Toggles socket level debugging (useful only to the *super*user*)*. The initial value for this toggle is FALSE.

options

Toggles the display of some internal *telnet* protocol processing (having to do with **TELNET** options). The initial value for this toggle is FALSE.

netdata

Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

?

Displays the legal **toggle** commands.

BUGS

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in ''line by line'' mode.

There is enough settable state to justify a *.telnetrc* file.

No capability for a *.telnetrc* file is provided.

In ''line by line'' mode, the terminal's *eof* character is only recognized (and sent to the remote system) when it is the first character on a line.

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

tftp − trivial file transfer program

SYNOPSIS

tftp [host]

DESCRIPTION

Tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case *tftp* uses *host* as the default host for future transfers (see the **connect** command below).

COMMANDS

Once *tftp* is running, it issues the prompt **tftp>** and recognizes the following commands:

connect *host-name* [*port*]
: Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections betweeen transfers; thus, the *connect* command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the *connect* command; the remote host can be specified as part of the *get* or *put* commands.

mode *transfer-mode*
: Set the mode for transfers; *transfer-mode* may be one of *ascii* or *binary*. The default is *ascii*.

put *file*

put *localfile remotefile*
put *file1 file2 ... fileN remote-directory*
: Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a *UNIX* machine.

get *filename*
get *remotename localname*
get *file1 file2 ... fileN*
: Get a file or set of files from the specified *sources*. *Source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

quit Exit *tftp*. An end of file also exits.

verbose Toggle verbose mode.

trace Toggle packet tracing.

status Show current status.

rexmt *retransmission-timeout*
: Set the per-packet retransmission timeout, in seconds.

timeout *total-transmission-timeout*
: Set the total transmission timeout, in seconds.

ascii Shorthand for "mode ascii"

binary Shorthand for "mode binary"

? [*command-name* ...]
: Print help information.

BUGS

Because there is no user-login or validation within the *TFTP* protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

NAME

time − time a command

SYNOPSIS

time command

DESCRIPTION

The given command is executed; after it is complete, *time* prints 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.

On a PDP-11, the execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

The times are printed on the diagnostic output stream.

Time is built in to *csh*(1), using a different output format.

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 100th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

Time is a built-in command to *csh*(1), with a much different syntax. This command is available as ''/bin/time'' to *csh* users.

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 cpu. It goes without saying that you must have a login 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*.

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. *Tip* prompts for the name of a local file to transmit.

~< Copy file from remote to local. *Tip* prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

~p *from* [*to*]
Send a file to a remote UNIX host. The put command causes the remote UNIX system to run 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.

~$ Pipe the output from a local UNIX process to the remote host. The command string sent to the local UNIX system is processed by the shell.

~# 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 with job control).

~^Y Stop only the ''local side'' of *tip* (only available with job control); the ''remote side'' of *tip*, the side that displays output from the remote host, is left running.

~? Get a summary of the tilde escapes

Tip uses the file /etc/remote to find how to reach a particular system and to find out how it should operate while talking to the system; refer to *remote*(5) 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, e.g. ''tip -300 mds''.

When *tip* establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in /etc/remote.

When *tip* prompts for an argument (e.g. 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.

Tip guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by *uucp*(1C).

During file transfers *tip* provides a running count of the number of lines transferred. When using the ~> and ~< commands, the ‘‘eofread’’ and ‘‘eofwrite’’ variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, ‘‘echocheck’’ may be set to indicate *tip* should synchronize with the remote system on the echo of each transmitted character.

When *tip* must dial a phone number to connect to a system it will print various messages indicating its actions. *Tip* supports the DEC DN-11 and Racal-Vadic 831 auto-call-units; the DEC DF02 and DF03, Ventel 212+, Racal-Vadic 3451, and Bizcomp 1031 and 1032 integral call unit/modems.

VARIABLES

Tip maintains a set of *variables* which control its operation. Some of these variable 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 concatenating 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. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values.

beautify
: (bool) Discard unprintable characters when a session is being scripted; abbreviated *be*.

baudrate
: (num) The baud rate at which the connection was established; abbreviated *ba*.

dialtimeout
: (num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated *dial*.

echocheck
: (bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is *off*.

eofread
: (str) The set of characters which signify and end-of-tranmission during a ~< file transfer command; abbreviated *eofr*.

eofwrite
: (str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated *eofw*.

eol
: (str) The set of characters which indicate an end-of-line. *Tip* will recognize escape characters only after an end-of-line.

escape
(char) The command prefix (escape) character; abbreviated *es*; default value is '~'.

exceptions
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated *ex*; default value is ''\t\n\f\b''.

force
(char) The character used to force literal data transmission; abbreviated *fo*; default value is '^P'.

framesize
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated *fr*.

host
(str) The name of the host to which you are connected; abbreviated *ho*.

prompt
(char) The character which indicates and end-of-line on the remote host; abbreviated *pr*; default value is '\n'. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on recipt of this character.

raise
(bool) Upper case mapping mode; abbreviated *ra*; default value is *off*. When this mode is enabled, all lower case letters will be mapped to upper case by *tip* for transmission to the remote machine.

raisechar
(char) The input character used to toggle upper case mapping mode; abbreviated *rc*; default value is '^A'.

record
(str) The name of the file in which a session script is recorded; abbreviated *rec*; default value is ''tip.record''.

script
(bool) Session scripting mode; abbreviated *sc*; default is *off*. When *script* is *true*, *tip* will record everything transmitted by the remote machine in the script record file specified in *record*. If the *beautify* switch is on, only printable ASCII characters will be included in the script file (those characters betwee 040 and 0177). The variable *exceptions* is used to indicate characters which are an exception to the normal beautification rules.

tabexpand
(bool) Expand tabs to spaces during file transfers; abbreviated *tab*; default value is *false*. Each tab is expanded to 8 spaces.

verbose
(bool) Verbose mode; abbreviated *verb*; default is *true*. When verbose mode is enabled, *tip* prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

SHELL
(str) The name of the shell to use for the ~! command; default value is ''/bin/sh'', or taken from the environment.

HOME
(str) The home directory to use for the ~c command; default value is taken from the environment.

FILES

/etc/remote	global system descriptions
/etc/phones	global phone number data base
${REMOTE}	private system descriptions
${PHONES}	private phone numbers
~/.tiprc	initialization file.
/usr/spool/uucp/LCK..*	lock file to avoid conflicts with *uucp*

DIAGNOSTICS

Diagnostics are, hopefully, self explanatory.

SEE ALSO

remote(5), phones(5)

BUGS

The full set of variables is undocumented and should, probably, be paired down.

NAME

tk − paginator for the Tektronix 4014

SYNOPSIS

tk [**−t**] [**−***N*] [**−p***L*] [file]

DESCRIPTION

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

The command line options are:

−t Don't wait between pages; for directing output into a file.

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

−p*L* Set page length to *L* lines.

SEE ALSO

pr(1)

NAME

tn3270 − full-screen remote login to IBM VM/CMS

SYNOPSIS

tn3270 sysname

DESCRIPTION

Tn3270 permits a full-screen, full-duplex connection from a VAX UNIX machine to an IBM machine running VM/CMS giving the appearance of being logged in directly to the remote machine on an IBM 3270 terminal. Of course you must have an account on the machine to which you wish to connect in order to log in. *Tn3270* looks to the user in many respects like the Yale ASCII Terminal Communication System II. *Tn3270* is actually a modification of the Arpanet TELNET user interface (see *telnet*(1)) that interprets and generates raw 3270 control streams.

Emulation of the 3270 terminal is done in the Unix process. This emulation involves mapping 3270-style commands from the host into appropriate sequences to control the user's terminal screen. *Tn3270* uses *curses*(3x) and the */etc/termcap* file to do this. The emulation also involves simulating the special 3270 keyboard keys (program function keys, etc.) by mapping sequences of keystrokes from the ASCII keyboard into appropriate 3270 control strings. This mapping is terminal dependent and is specified in a description file, */etc/map3270*, (see *map3270*(5)) or in an environment variable *MAP3270* (see *mset*(1)). Any special function keys on the ASCII keyboard are used whenever possible. If an entry for the user's terminal is not found, *tn3270* looks for an entry for the terminal type **unknown.** If this is not found, *tn3270* uses a default keyboard mapping (see *map3270*(5)).

The first character of each special keyboard mapping sequence is either an ASCII escape (ESC), a control character, or an ASCII delete (DEL). If the user types an unrecognized function key sequence, *tn3270* sends an ASCII bell (BEL), or a visual bell if defined in the user's termcap entry, to the user's terminal and nothing is sent to the IBM host.

If *tn3270* is invoked without specifying a remote host system name, it enters local command mode, indicated by the prompt ‘‘tn3270>’’. In this mode, *tn3270* accepts and executes the following commands:

open	connect to a remote host
close	close the current connection
quit	exit tn3270
z	suspend tn3270
status	print connection status
?	print help information

Other common *telnet* commands are not available in *tn3270. Tn3270* command mode may also be entered, after connecting to a host, by typing a special escape character (typically control-C).

While in command mode, any host login session is still alive but temporarily suspended. The host login session may be resumed by entering an empty line (press the RETURN key) in response to the command prompt. A session may be terminated by logging off the foreign host, or by typing ‘‘quit’’ or ‘‘close’’ while in local command mode.

FILES

/etc/termcap
/etc/map3270

AUTHOR

Greg Minshall

SEE ALSO

mset(1), telnet(1), termcap(3x), termcap(5), map3270(5), *Yale ASCII Terminal Communication System II Program Description/Operator's Manual* (IBM SB30-1911)

BUGS

Performance is slow and uses system resources prodigiously.

Not all 3270 functions are supported, nor all Yale enhancements.

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 a *file* exists, this is done by reading a character from the file and writing it back. If a *file* does not exist, an attempt will be made to create it unless the **−c** option is specified. The **−f** option will attempt to force the touch in spite of read and write permissions on a *file*.

SEE ALSO

utimes(2)

NAME

tp − manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or magtape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so './abc' can never be the same as '/usr/dmr/abc' even if '/usr/dmr' is the current directory. If no file argument is given, '**.**' is the default.

u updates the tape. **u** is like **r,** but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. **u** is the default command if none is given.

d deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.

x extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

m Specifies magtape as opposed to DECtape.

0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, **x** is default; for magtape '0' is the default.

v Normally *tp* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

c means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u.** This option is assumed with magtape since it is impossible to selectively overwrite magtape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option currently acts like **m**; *i.e.* **r** implies **c**, and neither **d** nor **u** are permitted.

w causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response **x** means 'exit'; the *tp* command terminates immediately. In the **x** function, files previously asked about have been extracted already. With **r, u,** and **d** no change has been made to the tape.

FILES

/dev/tap?
/dev/rmt?

SEE ALSO

ar(1), tar(1)

DIAGNOSTICS

Several; the non-obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

BUGS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by *tp* difficult to carry to other machines; *tar*(1) avoids the problem.

NAME

tr − translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **−cds** may be used: **−c** complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal; **−d** deletes all input characters in *string1;* **−s** squeezes all strings of repeated output characters that are in *string2* to single characters.

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.

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, nroff − text formatting and typesetting

SYNOPSIS

troff [option] ... [file] ...

nroff [option] ... [file] ...

DESCRIPTION

Troff formats text in the named *files* for printing on a Graphic Systems C/A/T phototypesetter; *nroff* is used for for typewriter-like devices. Their capabilities are described in the *Nroff/Troff user's manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (**−**) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

−o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N***−***M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end.

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline. *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

−m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

Troff only

−t Direct output to the standard output instead of the phototypesetter.

−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.

−a Send a printable ASCII approximation of the results to the standard output.

−pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

−F*fontdir*
The directory *fontdir* contains the font width tables instead of the default directory /usr/lib/fonts. This option can be used to produce output for devices besides the phototypesetter.

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

J. F. Ossanna, *Nroff/Troff user's manual*
B. W. Kernighan, *A TROFF Tutorial*
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 − terminal dependent initialization

SYNOPSIS

tset [options] [**−m** [ident][test baudrate]:type] ... [type]

reset [options] [**−m** [ident][test baudrate]:type] ... [type]

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. It 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/ttys*(5) 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 will be given an appropriate generic identifier such as *dialup*.

In the case where no arguments are specified, *tset* simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual concerns itself 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/ttys* 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 *termcap* 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 ''´'' characters; users of *csh*(1) must also put a ''\'' before any ''!'' used here.

Thus

    tset −m ´dialup>300:adm3a´ -m dialup:dw2 -m ´plugboard:?adm3a´

causes the terminal type to be set to an *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 (i.e. at 300 baud or less). (**NOTE:** 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.) 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 will be queried on a plugboard port as to whether they are 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 type found in the */etc/ttys* database will be taken to be 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 preferable to use the following command in your .login file to initialize the TERM and TERMCAP environment variables at the same time.

eval `tset -s *options...*`

It is also convenient to make an alias in your .cshrc:

alias tset ´eval `tset –s \!*`´

This allows the command:

tset 2621

to be invoked at any time to set the terminal and environment. **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.)

These commands cause *tset* to place the name of your terminal in the variable TERM in the environment; see *environ*(7).

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).

The options are:

–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 ^X (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.

–i*c* is similar to **–e** but for the interrupt character rather than the erase character; *c* defaults to ^C. 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 environment variable TERM.

–s Print the sequence of *csh* commands to initialize the environment variables TERM and TERMCAP based on the name of the terminal finally decided upon.

–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. See tty(4) for more detail.

–I suppresses transmitting terminal initialization strings.

–Q suppresses printing the ''Erase set to'' and ''Kill set to'' messages.

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. All arguments to *tset* may be used with reset.

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/ttys.

```
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 elses 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/ttys* 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/ttys* 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/ttys*. 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`
```

FILES

/etc/ttys port name to terminal type mapping database
/etc/termcap terminal capability database

SEE ALSO

csh(1), sh(1), stty(1), ttys(5), termcap(5), environ(7)

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.

This program can't intuit personal choices for erase, interrupt and line kill characters, so it leaves these set to the local system standards.

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)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

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) is given. In either case, the exit value is zero if the standard input is a terminal and one if it is not.

DIAGNOSTICS

'not a tty' if the standard input file is not a terminal.

NAME

ul − do underlining

SYNOPSIS

ul [**−i**] [**−t** *terminal*] [*name* ...]

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. The **−t** option overrides the terminal kind specified in the environment. The file */etc/termcap* is read 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.

The **−i** option causes *ul* to 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 outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

unifdef − remove ifdef'ed lines

SYNOPSIS

unifdef [**−t −l −c −D***sym* **−U***sym* **−id***sym* **−iu***sym*] ... [file]

DESCRIPTION

Unifdef is useful for removing ifdef'ed lines from a file while otherwise leaving the file alone. *Unifdef* is like a stripped-down C preprocessor: it is smart enough to deal with the nested ifdefs, comments, single and double quotes of C syntax so that it can do its job, but it doesn't do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined **−D***sym* or undefined **−U***sym* and the lines inside those ifdefs will be copied to the output or removed as appropriate. The ifdef, ifndef, else, and endif lines associated with *sym* will also be removed. Ifdefs involving symbols you don't specify are untouched and copied out along with their associated ifdef, else, and endif lines. If an ifdef X occurs nested inside another ifdef X, then the inside ifdef is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

The **−l** option causes *unifdef* to replace removed lines with blank lines instead of deleting them.

If you use ifdefs to delimit non-C lines, such as comments or code which is under construction, then you must tell *unifdef* which symbols are used for that purpose so that it won't try to parse for quotes and comments in those ifdef'ed lines. You specify that you want the lines inside certain ifdefs to be ignored but copied out with **−id***sym* and **−iu***sym* similar to **−D***sym* and **−U***sym* above.

If you want to use *unifdef* for plain text (not C code), use the **−t** option. This makes *unifdef* refrain from attempting to recognize comments and single and double quotes.

Unifdef copies its output to *stdout* and will take its input from *stdin* if no *file* argument is given. If the **−c** argument is specified, then the operation of *unifdef* is complemented, i.e. the lines that would have been removed or blanked are retained and vice versa.

SEE ALSO

diff(1)

DIAGNOSTICS

Premature EOF, inappropriate else or endif.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

BUGS

Does not know how to deal with *cpp* consructs such as

```
#if     defined(X) || defined(Y)
```

AUTHOR

Dave Yost

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). If the **−u** flag is used, just the lines that are not repeated in the original file are output. The **−d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **−u** and **−d** mode outputs.

The **−c** option supersedes **−u** and **−d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

NAME

units − conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

> *You have:* inch
> *You want:* cm
>   * *2.54000e+00*
>   / *3.93701e−01*

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

> *You have:* 15 pounds force/in2
> *You want:* atm
>   * *1.02069e+00*
>   / *9.79730e−01*

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

BUGS

Don't base your financial plans on the currency conversions.

NAME

uptime − show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w*(1) command.

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.

FILES

/etc/utmp

SEE ALSO

who(1)

NAME

uucp − unix to unix copy

SYNOPSIS

uucp [**−acCdfmr**] [**−n***user*] [**−g***grade*] [**−s***spool*] [**−x***debug*] source-file.... destination-file

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a pathname on your machine, or may have the form

system-name!pathname

where 'system-name' is taken from a list of system names that *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) a pathname prefixed by ~, where ~ is expanded into the system's public directory (usually /usr/spool/uucppublic);

(4) a partial pathname, which 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)).

The following options are interpreted by *uucp*.

−a Avoid doing a *getwd* to find the current directory. (This is sometimes used for efficiency.)

−c Use the source file when copying out rather than copying the file to the spool directory. (This is the default.)

−C Copy the source file to the spool directory and transmit the copy.

−d Make all necessary directories for the file copy. (This is the default.)

−f Do not make intermediate directories for the file copy.

−g*grade*
Grade is a single letter/number; lower ASCII sequence characters will cause a job to be transmitted earlier during a particular conversation. Default is 'n'. By way of comparison, *uux*(1C) defaults to 'A'; mail is usually sent at 'C'.

−m Send mail to the requester when the copy is complete.

−n*user* Notify *user* on remote system (i.e., send *user* mail) that a file was sent.

−r Do not start the transfer, just queue the job.

−s*spool* Use *spool* as the spool directory instead of the default.

−x*debug*
Turn on the debugging at level *debug.*

FILES

/usr/spool/uucp - spool directory
/usr/lib/uucp/* - other data and program files

SEE ALSO

uux(1C), mail(1)

D. A. Nowitz and M. E. Lesk, *A Dial-Up Network of UNIX Systems*.

D. A. Nowitz, *Uucp Implementation Description*.

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 the uucp administrator (usually UID 5).

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.)

At present *uucp* cannot copy to a system several "hops" away, that is, a command of the form

```
uucp myfile system1!system2!system3!yourfile
```

is not permitted. Use *uusend*(1C) instead.

When invoking *uucp* from *csh*(1), the '!' character must be prefixed by the '\' escape to inhibit *csh*'s history mechanism. (Quotes are not sufficient.)

Uucp refuses to copy a file that does not give read access to ''other''; that is, the file must have at least 0444 modes.

NAME

uuencode, uudecode − encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [source] remotedest **|** **mail** sys1!sys2!..!decode
uudecode [file]

DESCRIPTION

Uuencode and *uudecode* are used to send a binary file via uucp (or other) mail. This combination 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

atob(n), uusend(1C), uucp(1C), uux(1C), mail(1), uuencode(5)

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

uulog − display UUCP log files

SYNOPSIS

uulog [**−s** *sys*] [**−u** *user*]

DESCRIPTION

Uulog queries a log of *uucp*(1C) and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE.**

The options command *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/LOGFILE

SEE ALSO

uucp(1C), uux(1C).

NOTES

Very early releases of UUCP used separate log files for each of the UUCP utilities; *uulog* was used to merge the individual logs into a master file. This capability has not been necessary for some time and is no longer supported.

BUGS

UUCP's recording of which user issued a request is unreliable.

Uulog is little more than an overspecialized version of *grep*(1).

NAME

uuname − list names of UUCP hosts

SYNOPSIS

uuname [**−l**]

DESCRIPTION

Uuname lists the UUCP names of known systems. The **−l** option returns the local system name; this may differ from the *hostname*(1) for the system if the *hostname* is very long.

SEE ALSO

uucp(1C), uux(1C).

NAME

uuq − examine or manipulate the uucp queue

SYNOPSIS

uuq [**-l**] [**-h**] [**-s***system*] [**-u***user*] [**-d***jobno*] [**-r***sdir*] [**-b***baud*]

DESCRIPTION

Uuq is used to examine (and possibly delete) entries in the uucp queue.

When listing jobs, *uuq* uses a format reminiscent of *ls*. For the long format, information for each job listed includes job number, number of files to transfer, user who spooled the job, number of bytes to send, type of command requested (S for sending files, R for receiving files, X for remote uucp), and file or command desired.

Several options are available:

−h Print only the summary lines for each system. Summary lines give system name, number of jobs for the system, and total number of bytes to send.

−l Specifies a long format listing. The default is to list only the job numbers sorted across the page.

−s*system* Limit output to jobs for systems whose system names begin with *system*.

−u*user* Limit output to jobs for users whose login names begin with *user*.

−d*jobno* Delete job number *jobno* (as obtained from a previous *uuq* command) from the uucp queue. Only the UUCP Administrator is permitted to delete jobs.

−r*sdir* Look for files in the spooling directory *sdir* instead of the default directory.

−b*baud* Use *baud* to compute the transfer time instead of the default 1200 baud.

FILES

/usr/spool/uucp/ Default spool directory /usr/spool/uucp/C./C.* Control files /usr/spool/uucp/D*hostname*./D.* Outgoing data files /usr/spool/uucp/X./X.* Outgoing execution files

SEE ALSO

uucp(1C), uux(1C), uulog(1C), uusnap(8C)

BUGS

No information is available on work requested by the remote machine.

The user who requests a remote uucp command is unknown.

Uuq -l can be horrendously slow.

AUTHOR

Lou Salkind, New York University

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*(1) links needs to connect the two systems.

If the **−m** option is specified, the mode of the file on the remote end will be taken from the octal number given. Otherwise, the mode of the input file will be used.

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.

DIAGNOSTICS

If anything goes wrong any further away than the first system down the line, you will never hear about it.

SEE ALSO

uux(1), uucp(1), uuencode(1)

BUGS

This command should not 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 severly munged.

NAME

uux − unix to unix command execution

SYNOPSIS

uux [**-**] [**-cClLnprz**] [**-a**_name_] [**-g**_grade_] [**-x**_debug_] command-string

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

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 path name;

(2) a path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name prefixed by **~;** where **~** is expanded to the system's public directory (usually /usr/spool/uucppublic);

(4) a partial pathname, which is prefixed by the current directory.

As an example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

will get the **file1** and **file2** files from the ''usg'' and ''pwba'' machines, execute a *diff*(1) command and put the results in **file.diff** in the local /usr/spool/uucppublic/dan/ directory.

Any special shell characters, such as **<>;|**, should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

Uux will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!wc b!/usr/file1 \(c!/usr/file2 \)
```

get **/usr/file1** from system ''b'' and send it to system ''a'', perform a *wc* command on that file and send the result of the *wc* command to system ''c''.

Uux will notify you by mail if the requested command on the remote system was disallowed. This notification can be turned off by the **−n** option.

The following *options* are interpreted by *uux*:

− The standard input to *uux* is made the standard input to the *command-string*.

−a_name_ Use *name* as the user identification replacing the initiator user-id.

−c Do not copy local file to the spool directory for transfer to the remote machine (this is the default).

−C Force the copy of local files to the spool directory for transfer.

−g_grade_

Grade is a single letter/number, from **0** to **9**, **A** to **Z**, or **a** to **z**; **0** is the highest, and **z** is the lowest grade. The default is **A**; by comparison *uucp*(1C) defaults to **n** and mail is usually sent at grade **C**. Lower grades should be specified for high-volume jobs, such as news.

−l Try and make a link from the original file to the spool directory. If the link cannot be made, copy the file.

−n Do not notify the user when the command completes.

−p Same as −: The standard input to *uux* is made the standard input to the *command-string*.

−r Do not start the file transfer, just queue the job.

−x*debug*
Produce debugging output on stdout. The debug is a number between 0 and 9; higher numbers give more detailed information. Debugging is permitted only for privileged users (specifically, those with read access to *L.sys*(5).

−z Notify the user only if the command fails.

−L Start up *uucico* with the **-L** flag. This will force calls to be made to local sites only (see *uucico*(8C)).

FILES

/usr/spool/uucp	spool directories
/usr/lib/uucp/∗	UUCP configuration data and daemons

SEE ALSO

uucp(1C), uucico(8C), uuxqt(8C).

WARNING

For security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail*(1)) 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.

When invoking *uux* from *csh*(1), the '!' character must be prefixed by the '\' escape to inhibit *csh*'s history mechanism. (Quotes are not sufficient.)

NAME

vacation − return ‘‘I am on vacation’’ indication

SYNOPSIS

vacation −I
vacation user

DESCRIPTION

Vacation returns a message to the sender of a message telling that you are on vacation. The intended use is in a *.forward* file. For example, your *.forward* file might have:

```
\eric, "|vacation eric"
```

which would send messages to you (assuming your login name was eric) and send a message back to the sender.

Vacation expects a file *.vacation.msg* in your home directory containing a message to be sent back to each sender. It should be an entire message (including headers). For example, it might say:

```
From: eric@ucbmonet.Berkeley.EDU (Eric Allman)
Subject: I am on vacation
Delivered-By-The-Graces-Of: the Vacation program

I am on vacation until July 22. If you have something urgent,
please contact Joe Kalash <kalash@ucbingres.Berkeley.EDU>.
        --eric
```

This message will only be sent once a week to each unique sender. The people who have sent you messages are kept in the files *.vacation.pag* and *.vacation.dir* in your home directory. The **−I** option initializes these files, and should be executed before you modify your *.forward* file.

If the **−I** flag is not specified, *vacation* reads the first line from the standard input for a UNIX-style ‘‘From’’ line to determine the sender. If this is not present, a nasty diagnostic is produced. *Sendmail*(8) includes the ‘‘From’’ line automatically.

No message is sent if the initial ‘‘From’’ line includes the string ‘‘-REQUEST@’’ or if a ‘‘Precedence: bulk’’ or ‘‘Precedence: junk’’ line is included in the header.

SEE ALSO

sendmail(8)

NAME

vgrind − grind nice listings of programs

SYNOPSIS

vgrind [**−f**] [**−**] [**−t**] [**−n**] [**−x**] [**−W**] [**−sn**] [**−h** header] [**−d** file] [**−l***language*] name ...

DESCRIPTION

Vgrind formats the program sources which are 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.

The options are:

−f forces filter mode

− forces input to be taken from standard input (default if **−f** is specified)

−t similar to the same option in *troff* causing formatted text to go to the standard output

−n forces no keyword bolding

−x outputs 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 forces 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 .ps)

−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**), MODEL (**−lm**),**C** (**−lc** or the default), CSH (**−lcsh**), SHELL (**−lsh**), RATFOR (**−lr**), MODULA2 (**−lmod2**), YACC (**−lyacc**), ISP (**−lisp**), 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

AUTHOR

Dave Presotto & William Joy

SEE ALSO

vlp(1), vtroff(1), vgrindefs(5)

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.*

For **MODEL** – function names need to appear on the same line as the keywords *is beginproc.*

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 in recognizing functions should be used here.

Filter mode does not work in documents using the −me or −ms macros. (So what use is it anyway?)

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* run the same code; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

The *Vi Quick Reference* card and the *Introduction to Display Editing with Vi* provide full details on using *vi.*

FILES

See *ex*(1).

SEE ALSO

ex (1), edit (1), ''Vi Quick Reference'' card, ''An Introduction to Display Editing with Vi''.

AUTHOR

William Joy
Mark Horton added macros to *visual* mode and is maintaining version 3

BUGS

Software tabs using **^T** work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open.*

NAME

vlp − Format Lisp programs to be printed with nroff, vtroff, or troff

SYNOPSIS

vlp [**−p** *pointsize*] [**−d**] [**−f**] [**−l**] [**−v**] [**−T** *title1*] file1 [**−T** *title2*] file2 ...

DESCRIPTION

Vlp formats the named files so that they can be run through nroff, vtroff, or troff to produce listings that line-up and are attractive. The first non-blank character of each line is lined-up vertically, as in the source file. Comments (text beginning with a semicolon) are printed in italics. Each function's name is printed in bold face next to the function. This format makes Lisp code look attractive when it is printed with a variable width font.

Normally, *vlp* works as a filter and sends its output to the standard output. However, the **−v** switch pipes the output directly to vtroff. If no files are specified, then *vlp* reads from the standard input.

The following options are available:

−p The **−p** switch changes the size of the text from its default value of 8 points to one of 6, 8, 10, or 12 points. Once set, the point size is used for all subsequent files. This point size does not apply to embedded text (see −*f* below).

−d The **−d** switch puts *vlp* into debugging mode.

−f *Vlp* has a filtered mode in which all lines are passed unmodified, except those lines between the directives **.Ls** and **.Le**. This mode can be used to format Lisp code that is embedded in a document. The directive **.Ls** takes an optional argument that gives the point size for the embedded code. If not size is specified, the size of the surrounding text is used.

−l The **−l** switch prevents *vlp* from placing labels next to functions. This switch is useful for embedded Lisp code, where the labels would be distracting.

−v This switch cause *vlp* to send its output to vtroff rather than the standard output.

−T A title to be printed on each page may be specified by using the **−T** switch. The **−T** switch applies only to the next file name given. Titles are not printed for embedded text (see **−f**, above). This switch may not be used if *vlp* is reading from the standard input.

FILES

/usr/lib/vlpmacs troff/nroff macros

AUTHOR

Originally written by John K. Foderaro, with additional changes by Kevin Layer and James Larus.

SEE ALSO

vgrind(1), lisp(1)

BUGS

vlp transforms \ into \\ so that it will be printed out. Hence, troff commands cannot be embedded in Lisp code.

NAME

vmstat − report virtual memory statistics

SYNOPSIS

vmstat [**−fsi**] [drives] [interval [count]]

DESCRIPTION

Vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and cpu activity. If given a **−f** argument, it instead reports on the number of *forks* and *vforks* since system startup and the number of pages of virtual memory involved in each kind of fork. If given a **−s** argument, it instead prints the contents of the *sum* structure, giving the total number of several kinds of paging related events which have occurred since boot. If given a **−i** argument, it instead reports on the number of *interrupts* taken by each device since system startup.

If none of these options are given, *vmstat* will report in the first line a summary of the virtual memory activity since the system has been booted. If *interval* is specified, then successive lines are summaries over the last *interval* seconds. ''vmstat 5'' will print what the system is doing every five seconds; this is a good choice of printing interval since this is how often some of the statistics are sampled in the system; others vary every second, running the output for a while will make it apparent which are recomputed every second. If a *count* is given, the statistics are repeated *count* times. The format fields are:

Procs: information about numbers of processes in various states.

r	in run queue
b	blocked for resources (i/o, paging, etc.)
w	runnable or short sleeper (< 20 secs) but swapped

Memory: information about the usage of virtual and real memory. Virtual pages are considered active if they belong to processes which are running or have run in the last 20 seconds. A ''page'' here is 1024 bytes.

avm	active virtual pages
fre	size of the free list

Page: information about page faults and paging activity. These are averaged each five seconds, and given in units per second.

re	page reclaims (simulating reference bits)
at	pages attached (found in free list)
pi	pages paged in
po	pages paged out
fr	pages freed per second
de	anticipated short term memory shortfall
sr	pages scanned by clock algorithm, per-second

up/hp/rk/ra: Disk operations per second (this field is system dependent). Typically paging will be split across several of the available drives. The number under each of these is the unit number.

Faults: trap/interrupt rate averages per second over last 5 seconds.

in	(non clock) device interrupts per second
sy	system calls per second
cs	cpu context switch rate (switches/sec)

Cpu: breakdown of percentage usage of CPU time

us	user time for normal and low priority processes
sy	system time
id	cpu idle

If more than 4 disk drives are configured in the system, *vmstat* displays only the first 4 drives, with priority given to Massbus disk drives (i.e. if both Unibus and Massbus drives are present and the total number of drives exceeds 4, then some number of Unibus drives will not be displayed in favor of the Massbus drives). To force *vmstat* to display specific drives, their names may be supplied on the command line.

FILES

/dev/kmem, /vmunix

SEE ALSO

systat(1), *iostat*(1)

The sections starting with ''Interpreting system activity'' in *Installing and Operating 4.2bsd*.

NAME

vnews − read news articles

SYNOPSIS

vnews [**−a** *date*] [**−n** *newsgroups*] [**−t** *titles*] [**−rxuc**]

vnews −s

DESCRIPTION

Vnews is a program for reading USENET news. It is based on *readnews*(1) but has a CRT oriented interface. The list of available commands is quite similar, although since *vnews* is a ''visual'' interface, most *vnews* commands do not have to be terminated by a newline.

Vnews uses all but the last two lines of the screen to display the current article. The next-to-last line is the secondary prompt line, and is used to input string arguments to commands. The last line contains several fields. The first field is the prompt field. If *vnews* is at the end of an article, the prompt is ''next?''; otherwise the prompt is ''more?''. The second field is the newsgroup field, which displays the current newsgroup, the number of the current article, and the number of the last article in the newsgroup. The third field contains the current time, and the last field contains the word ''mail'' if you have mail. When you receive new mail, the bell on the terminal is rung and the word ''MAIL'' appears in capital letters for 30 seconds.

Vnews without any arguments prints unread articles.

The following flags determine the selection of articles.

−a [*date*] Select articles posted after the given *date* (in *getdate*(3) format).

−n *newsgroups*
Select articles belonging to *newsgroups.*

−t *titles* Select articles whose titles contain one of the strings specified by *titles.*

−r Print the articles in reverse order.

−x Ignore *.newsrc* file. That is, select articles that have already been read as well as new ones.

−u Update the *.newsrc* file every 5 minutes, as in the case of an unreliable system. (Note that if the *.newsrc* file is updated, the **x** command will not restore it to its original contents.)

If the **−c** flag is specified, *vnews* will print the first page of the article, instead of just the header.

You can use the **−s** flag to print the newsgroup subscription list.

Vnews maintains a *.newsrc* file in the your home directory that specifies all news articles already read. It is updated at the end of each reading session unless the **−x** option was specified. If the environment variable **NEWSRC** is present, it should be the path name of a file to be used in place of *.newsrc*.

If you wish, an options line may be placed in your *.newsrc* file. This line starts with the word **options** (left justified) followed by the list of standard options just as they would be typed on the command line. Such a list may include: the **−n** flag along with a newsgroup list and/or the **−r** or **−t** flag. Continuation lines begin with a space or tab character.

ENVIRONMENT

Options can be specified in the **NEWSOPTS** environment parameter. Where conflicts exist, options on the command line take precedence, followed by the *.newsrc* **options** line, and lastly the **NEWSOPTS** parameter.

When the user uses the reply command, the environment parameter **MAILER** will be used to determine which mailer to use. The default is usually */bin/mail*.

If the user so desires, he may specify a specific paging program for articles. The environment parameter **PAGER** should be set to the paging program. The name of the article is referenced with a ''%'', as in the **−c** option. If no ''%'' is present, the article will be piped to the program. Paging may be disabled by setting **PAGER** to a null value.

If **EDITOR** is set, it will be used in place of the default editor on your system to edit replies and follow-ups.

If **NAME** is set, it will be used as your full name when posting news or submitting a follow-up. If it is not set, the name will be taken from the file *.name* in your home directory. If this file is not present, the name will be taken from */etc/passwd*.

If **NEWSARCHIVE** is set, a copy of any articles you post or follow-up to, will be saved in the specified file. If it is the null string, they will be copied in *author_copy* in your home directory.

If **NEWSBOX** is set, the filename you specify when you save or write a file will be prepended with **NEWSBOX** unless the filename is an absolute pathname.

If **NEWSRC** is set, it will be used in place of the *.newsrc* file in your home directory.

If **ORGANIZATION** is set, it will be used as the name of your organization whenever you post an article. The default is compiled in and is usually correct. Typically, you would only use this if you were reading news at a site other than normal. (Or if you are trying to be cute.)

COMMANDS

Each *vnews* command may be preceded by a count. Some commands use the count; others ignore it. If count is omitted, it defaults to one. Some commands prompt for an argument on the second line from the bottom of the screen. Standard UNIX erase and kill processing is done on this argument. The argument is terminated by a return. An interrupt (DEL or BREAK) gets you out of any partially entered command.

In the following table, ^B is used as a shorthand for Control-B.

Command	Meaning
CR	A carriage return prints more of the current article, or goes on to the next article if you are at the end of the current article. A **SPACE** is equivalent to **CR**.
^B	Go backwards *count* pages.
^F	Go forward *count* pages.
^D	Go forwards half a page.
^U	Go backwards half a page.
^Z	Go forwards *count* lines.
^E	Go backwards *count* lines.
^L	Redraw the screen. **^L** may be typed at any time.
b	Back up one article in the current group.
c	Cancel the article. Only the author of the article or the super user can do this.
d	Read a digest. Breaks up a digest into separate articles and permits you to read and reply to each piece.
e	Erase. Forget that this article was read.
f	Submit a follow-up article. You will be placed in your **EDITOR** to compose the text of the follow-up.
h	Go back to the top of the article and display only the header.
l	Redisplay the article after you have sent a follow-up or reply.
m	Move on to the next item in a digest.
n	No. Go on to the next article without printing current one. **.** is equivalent to **n**. This is convenient if your terminal has a keypad.

p Show the parent article (the article that the current article is a follow-up to). This doesn't work if the current article was posted by A-news or notesfiles. To switch between the current and parent articles, use the **–** command. Unfortunately, if you use several **p** commands to trace the discussion back further, there is no command to return to the original level.

q Quit. The *.newsrc* file will be updated unless **–x** was on the command line.

r Reply. Reply to article's author via mail. You are placed in your **EDITOR** with a header specifying ''To'', ''Subject'', and ''References'' lines taken from the message. You may change or add headers, as appropriate. 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.

R This is the same as **r** except the body of the article is included in your mail message for you.

ESC-r Reply directly. You are placed in your **MAILER** as if you had run it specifying the author of the article as the recipient of a letter.

s [*file*] Save. The article is appended to the named file. The default is *Articles*. If the first character of the file name is ''|'', the rest of the file name 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 file name is ''/'', it is taken as the full pathname of a file. If **NEWSBOX** (in the environment) is set to a full pathname, and the file contains no ''/'', the file is saved in **NEWSBOX**. Otherwise, it is saved relative to **HOME**.

ug Unsubscribe to the current group. This is a two character command to ensure that it is not typed accidentally and to leave room for other types of unsubscribes (e.g. unsubscribe to discussion).

v Print the current version of the news software.

w Write. Like save **s**, except that the headers are not written out.

x Exit. Like quit except that *.newsrc* is not updated.

y Yes. Print the current article and go to the next.

[*n*]**A** Go to article number *n* in the current newsgroup.

D Decrypts a joke. It only handles rot 13 jokes. The **D** command is a toggle; typing another **D** re-encrypts the joke.

H Print a very verbose header, containing all known information about the article.

K Kill (mark as read) the rest of the articles in the current group. This is useful if you can't keep up with the volume in the newsgroup, but don't want to unsubscribe.

N [*newsgroup*]
Go to the next newsgroup or named newsgroup.

[*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.

< Prompt for an article ID or the rest of a message ID. It will display the article if it exists.

Report the name and size of the newsgroup.

? Print an short help message.

! Passes the rest of the command line to the shell. The environment variable **A** is set to the name of the file containing the current article. If the last character of the command is a **&**, then the **&** is deleted and the command is run in the background with stdin, stdout and stderr redirected to */dev/null*. If the command is missing, the shell is invoked. Use the **l** command (or essentially any other command) to turn on the display after the program terminates.

EXAMPLES

vnews Read all unread articles using the *visual* interface. The *.newsrc* file is updated at the end of the session.

vnews −n all !mod.all −r
Read all unread articles except articles whose newsgroups begin with **mod.** in reverse order. The *.newsrc* file is updated at the end of the session.

vnews −n all −a last thursday
Print every unread article since last Thursday. The *.newsrc* file is updated at the end of the session.

vnews −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
/usr/lib/news/vnews.help	Help file for *visual* interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

checknews(1N), inews(1N), postnews(1N), readnews(1N), getdate(3N), news(5N), newsrc(5N), expire(8N), recnews(8N), sendnews(8N), uurec(8N), (many of these not in printed manuals, but available online)
"How to Read the Network News" (USD:9)

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.) Troff should look directly in the font file but it doesn't.

Vwidth should be used after editing a font with *fed(1).* It is not necessary to use vwidth unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use vwidth if the physical width of the glyph (e.g. the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and vwidth run.

Vwidth produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the .o file) saved. The resulting file should be placed in /usr/lib/font in the file ft*xx* where is a one or two letter code that is the logical (internal to troff) font name. This name can be found by looking in the file /usr/lib/fontinfo/*fname** where *fname* is the external name of the font.

SEE ALSO

fed(1), vfont(5), troff(1), vtroff(1)

BUGS

Produces the C file using obsolete syntax that the portable C compiler complains about.

NAME

w − who is on and what they are doing

SYNOPSIS

w [**−h**] [**−s**] [user]

DESCRIPTION

W prints 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 output 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.

The **−h** flag suppresses the heading. The **−s** flag asks for 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** gives the long output, which is the default.

If a *user* name is included, the output will be restricted to that user.

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), finger(1), ps(1)

AUTHOR

Mark Horton

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 detection of 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 **&** have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the Shell itself executes *wait,* without creating a new process.

SEE ALSO

sh(1)

BUGS

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

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)

DIAGNOSTICS

'Cannot send to ...' when the open on a user's tty file fails.

NAME

wc − word count

SYNOPSIS

wc [**−lwc**] [name ...]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If an argument beginning with one of ''lwc'' is present, the specified counts (lines, words, or characters) are selected by the letters **l**, **w**, or **c**. The default is **−lwc**.

BUGS

NAME

what − show what versions of object modules were used to construct a file

SYNOPSIS

what name ...

DESCRIPTION

What reads each file and searches for sequences of the form ‘‘@(#)’’ as inserted by the source code control system. It then prints the remainder of the string after this marker, up to a null character, newline, double quote, or ‘‘>’’ character.

BUGS

As SCCS is not licensed with UNIX/32V, this is a rewrite of the *what* command which is part of SCCS, and may not behave exactly the same as that command does.

NAME

whatis − describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

Whatis looks up a given command and gives 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.

Whatis is actually just the **−f** option to the *man*(1) command.

FILES

/usr/man/whatis Data base

SEE ALSO

man(1), catman(8)

NAME

whereis − locate source, binary, and or manual for program

SYNOPSIS

whereis [**−sbm**] [**−u**] [**−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’’, e.g. ‘‘.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. If any of the **−b, −s** or **−m** flags are given then *whereis* searches only for binaries, sources or manual sections respectively (or any two thereof). The **−u** flag may be used to 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.

Finally, the **−B −M** and **−S** flags may be used to change or otherwise limit the places where *whereis* searches. The **−f** file flags is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
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 the program uses *chdir*(2) to run faster, pathnames given with the **−M −S** and **−B** must be full; i.e. 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

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

Who, without an argument, lists the login name, terminal name, and login time for each current UNIX user.

Without an argument, *who* examines the */etc/utmp* file to obtain its information. If a file is given, that file is examined. Typically the given file will be /usr/adm/wtmp, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with '/dev/' suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in 'who am I' (and also 'who are you'), *who* tells who you are logged in as.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5)

NAME

whoami − print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES

/etc/passwd Name data base

SEE ALSO

who (1)

NAME

whois − DARPA Internet user name directory service

SYNOPSIS

whois name

DESCRIPTION

whois help

Produces a helpful message similar to the following:

Please enter a name or a handle (‘‘ident"), such as ‘‘Smith" or ‘‘SRI-NIC". Starting with a period forces a name-only search; starting with exclamation point forces handle-only. Examples:

Smith	[looks for name or handle SMITH]
!SRI-NIC	[looks for handle SRI-NIC only]
.Smith, John	[looks for name JOHN SMITH only]

Adding ‘‘..." to the argument will match anything from that point, e.g. ‘‘ZU..." will match ZUL, ZUM, etc.

To have the ENTIRE membership list of a group or organization, if you are asking about a group or org, shown with the record, use an asterisk character ‘*’ directly preceding the given argument. [CAUTION: If there are a lot of members this will take a long time!] You may of course use exclamation point and asterisk, or a period and asterisk together.

SEE ALSO

RFC 812: Nicname/Whois

NAME

window − window environment

SYNOPSIS

window [**−t**] [**−f**] [**−d**] [**−e escape-char**] [**−c command**]

DESCRIPTION

Window implements a window environment on ASCII terminals.

A window is a rectangular portion of the physical terminal screen associated with a set of processes. Its size and position can be changed by the user at any time. Processes communicate with their window in the same way they normally interact with a terminal--through their standard input, output, and diagnostic file descriptors. The window program handles the details of redirecting input an output to and from the windows. At any one time, only one window can receive input from the keyboard, but all windows can simultaneously send output to the display.

Windows can overlap and are framed as necessary. Each window is named by one of the digits ''1'' to ''9''. This one character identifier, as well as a user definable label string, are displayed with the window on the top edge of its frame. A window can be designated to be in the *foreground*, in which case it will always be on top of all normal, non-foreground windows, and can be covered only by other foreground windows. A window need not be completely within the edges of the terminal screen. Thus a large window (possibly larger than the screen) may be positioned to show only a portion of its full size.

Each window has a cursor and a set of control functions. Most intelligent terminal operations such as line and character deletion and insertion are supported. Display modes such as underlining and reverse video are available if they are supported by the terminal. In addition, similar to terminals with multiple pages of memory, each window has a text buffer which can have more lines than the window itself.

OPTIONS

When *window* starts up, the commands (see long commands below) contained in the file *.windowrc* in the user's home directory are executed. If it does not exist, two equal sized windows spanning the terminal screen are created by default.

The command line options are

−t Turn on terse mode (see *terse* command below).

−f Fast. Don't perform any startup action.

−d Ignore *.windowrc* and create the two default windows instead.

−e escape-char
Set the escape character to *escape-char*. *Escape-char* can be a single character, or in the form ^*X* where *X* is any character, meaning control-*X*.

−c command
Execute the string *command* as a long command (see below) before doing anything else.

PROCESS ENVIRONMENT

With each newly created window, a shell program is spawned with its process environment tailored to that window. Its standard input, output, and diagnostic file descriptors are bound to one end of either a pseudo-terminal (*pty* (4)) or a UNIX domain socket (*socketpair* (4)). If a pseudo-terminal is used, then its special characters and modes (see *stty* (1)) are copied from the physical terminal. A *termcap* (5) entry tailored to this window is created and passed as environment (*environ* (5)) variable *TERMCAP*. The termcap entry contains the window's size and characteristics as well as information from the physical terminal, such as the existence of underline, reverse video, and other display modes, and the codes produced by the terminal's function keys, if any. In addition, the window size attributes of the pseudo-terminal are set to reflect the size of this window, and updated whenever it is changed by the user. In particular, the editor *vi* (1) uses this information to redraw its display.

OPERATION

During normal execution, *window* can be in one of two states: conversation mode and command mode. In conversation mode, the terminal's real cursor is placed at the cursor position of a particular window--called the current window--and input from the keyboard is sent to the process in that window. The current window is always on top of all other windows, except those in foreground. In addition, it is set apart by highlighting its identifier and label in reverse video.

Typing *window*'s escape character (normally ^P) in conversation mode switches it into command mode. In command mode, the top line of the terminal screen becomes the command prompt window, and *window* interprets input from the keyboard as commands to manipulate windows.

There are two types of commands: short commands are usually one or two key strokes; long commands are strings either typed by the user in the command window (see the ``:'' command below), or read from a file (see *source* below).

SHORT COMMANDS

Below, # represents one of the digits ``1'' to ``9'' corresponding to the windows 1 to 9. *^X* means control-*X*, where *X* is any character. In particular, ^^ is control-^. *Escape* is the escape key, or *^[*.

Select window # as the current window and return to conversation mode.

%# Select window # but stay in command mode.

^^ Select the previous window and return to conversation mode. This is useful for toggling between two windows.

escape Return to conversation mode.

^P Return to conversation mode and write ^P to the current window. Thus, typing two ^P's in conversation mode sends one to the current window. If the *window* escape is changed to some other character, that character takes the place of ^P here.

? List a short summary of commands.

^L Redraw the screen.

q Exit *window*. Confirmation is requested.

^Z Suspend *window*.

w Create a new window. The user is prompted for the positions of the upper left and lower right corners of the window. The cursor is placed on the screen and the keys ``h'', ``j'', ``k'', and ``l'' move the cursor left, down, up, and right, respectively. The keys ``H'', ``J'', ``K'', and ``L'' move the cursor to the respective limits of the screen. Typing a number before the movement keys repeats the movement that number of times. Return enters the cursor position as the upper left corner of the window. The lower right corner is entered in the same manner. During this process, the placement of the new window is indicated by a rectangular box drawn on the screen, corresponding to where the new window will be framed. Typing escape at any point cancels this command.

This window becomes the current window, and is given the first available ID. The default buffer size is used (see *nline* command below).

Only fully visible windows can be created this way.

c# Close window #. The process in the window is sent the hangup signal (see *kill* (1)). *Csh* (1) should handle this signal correctly and cause no problems.

m# Move window # to another location. A box in the shape of the window is drawn on the screen to indicate the new position of the window, and the same keys as those for the *w* command are used to position the box. The window can be moved partially off-screen.

M# Move window # to its previous position.

s# Change the size of window *#*. The user is prompted to enter the new lower right corner of the window. A box is drawn to indicate the new window size. The same keys used in *w* and *m* are used to enter the position.

S# Change window # to its previous size.

^Y Scroll the current window up by one line.

^E Scroll the current window down by one line.

^U Scroll the current window up by half the window size.

^D Scroll the current window down by half the window size.

^B Scroll the current window up by the full window size.

^F Scroll the current window down by the full window size.

h Move the cursor of the current window left by one column.

j Move the cursor of the current window down by one line.

k Move the cursor of the current window up by one line.

l Move the cursor of the current window right by one column.

^S Stop output in the current window.

^Q Start output in the current window.

: Enter a line to be executed as long commands. Normal line editing characters (erase character, erase word, erase line) are supported.

LONG COMMANDS

Long commands are a sequence of statements parsed much like a programming language, with a syntax similar to that of C. Numeric and string expressions and variables are supported, as well as conditional statements.

There are two data types: string and number. A string is a sequence of letters or digits beginning with a letter. ‘‘_’’ and ‘‘.’’ are considered letters. Alternately, non-alphanumeric characters can be included in strings by quoting them in ‘‘"’’ or escaping them with ‘‘\’’. In addition, the ‘‘\’’ sequences of C are supported, both inside and outside quotes (e.g., ‘‘\n’’ is a new line, ‘‘\r’’ a carriage return). For example, these are legal strings: abcde01234, "&#$^*&#", ab"$#"cd, ab\$\#cd, "/usr/ucb/window".

A number is an integer value in one of three forms: a decimal number, an octal number preceded by ‘‘0’’, or a hexadecimal number preceded by ‘‘0x’’ or ‘‘0X’’. The natural machine integer size is used (i.e., the signed integer type of the C compiler). As in C, a non-zero number represents a boolean true.

The character ‘‘#’’ begins a comment which terminates at the end of the line.

A statement is either a conditional or an expression. Expression statements are terminated with a new line or ‘‘;’’. To continue an expression on the next line, terminate the first line with ‘‘\’’.

CONDITIONAL STATEMENT

Window has a single control structure: the fully bracketed if statement in the form

```
if <expr> then
        <statement>
        . . .
elsif <expr> then
        <statement>
        . . .
else
        <statement>
        . . .
endif
```

The *else* and *elsif* parts are optional, and the latter can be repeated any number of times. *<Expr>* must be numeric.

EXPRESSIONS

Expressions in *window* are similar to those in the C language, with most C operators supported on numeric operands. In addition, some are overloaded to operate on strings.

When an expression is used as a statement, its value is discarded after evaluation. Therefore, only expressions with side effects (assignments and function calls) are useful as statements.

Single valued (no arrays) variables are supported, of both numeric and string values. Some variables are predefined. They are listed below.

The operators in order of increasing precedence:

<expr1> = <expr2>
: Assignment. The variable of name *<expr1>*, which must be string valued, is assigned the result of *<expr2>*. Returns the value of *<expr2>*.

<expr1> ? <expr2> : <expr3>
: Returns the value of *<expr2>* if *<expr1>* evaluates true (non-zero numeric value); returns the value of *<expr3>* otherwise. Only one of *<expr2>* and *<expr3>* is evaluated. *<Expr1>* must be numeric.

<expr1> || <expr2>
: Logical or. Numeric values only. Short circuit evaluation is supported (i.e., if *<expr1>* evaluates true, then *<expr2>* is not evaluated).

<expr1> && <expr2>
: Logical and with short circuit evaluation. Numeric values only.

<expr1> | <expr2>
: Bitwise or. Numeric values only.

<expr1> ^ <expr2>
: Bitwise exclusive or. Numeric values only.

<expr1> & <expr2>
: Bitwise and. Numeric values only.

<expr1> == <expr2>, <expr1> != <expr2>
: Comparison (equal and not equal, respectively). The boolean result (either 1 or 0) of the comparison is returned. The operands can be numeric or string valued. One string operand forces the other to be converted to a string in necessary.

<expr1> < <expr2>, <expr1> > <expr2>,
: Less than, greater than, less than or equal to, greater than or equal to. Both numeric and string values, with automatic conversion as above.

<expr1> << <expr2>, <expr1> >> <expr2>
: If both operands are numbers, *<expr1>* is bit shifted left (or right) by *<expr2>* bits. If *<expr1>* is a string, then its first (or last) *<expr2>* characters are returns (if *<expr2>* is also a string, then its length is used in place of its value).

<expr1> + <expr2>, <expr1> - <expr2>
: Addition and subtraction on numbers. For ''+'', if one argument is a string, then the other is converted to a string, and the result is the concatenation of the two strings.

<expr1> * <expr2>, <expr1> / <expr2>,
: Multiplication, division, modulo. Numbers only.

-<expr>, ~<expr>, !<expr>, $<expr>, $?<expr>
: The first three are unary minus, bitwise complement and logical complement on numbers only. The operator, ''$'', takes *<expr>* and returns the value of the variable of that name. If *<expr>*

is numeric with value *n* and it appears within an alias macro (see below), then it refers to the nth argument of the alias invocation. ''$?'' tests for the existence of the variable *<expr>*, and returns 1 if it exists or 0 otherwise.

<expr>(<arglist>)

Function call. *<Expr>* must be a string that is the unique prefix of the name of a builtin *window* function or the full name of a user defined alias macro. In the case of a builtin function, *<arglist>* can be in one of two forms:

<expr1>, <expr2>, . . .
argname1 = <expr1>, argname2 = <expr2>, . . .

The two forms can in fact be intermixed, but the result is unpredictable. Most arguments can be omitted; default values will be supplied for them. The *argnames* can be unique prefixes of the the argument names. The commas separating arguments are used only to disambiguate, and can usually be omitted.

Only the first argument form is valid for user defined aliases. Aliases are defined using the *alias* builtin function (see below). Arguments are accessed via a variant of the variable mechanism (see ''$'' operator above).

Most functions return value, but some are used for side effect only and so must be used as statements. When a function or an alias is used as a statement, the parenthesis surrounding the argument list may be omitted. Aliases return no value.

BUILTIN FUNCTIONS

The arguments are listed by name in their natural order. Optional arguments are in square brackets (''[ ]''). Arguments that have no names are in angle brackets (''<>'').

alias([<string>], [<string-list>])

If no argument is given, all currently defined alias macros are listed. Otherwise, *<string>* is defined as an alias, with expansion *<string-list>*. The previous definition of *<string>*, if any, is returned. Default for *<string-list>* is no change.

close(<window-list>)

Close the windows specified in *<window-list>*. If *<window-list>* is the word *all*, than all windows are closed. No value is returned.

cursormodes([modes])

Set the window cursor to *modes*. *Modes* is the bitwise or of the mode bits defined as the variables *m_ul* (underline), *m_rev* (reverse video), *m_blk* (blinking), and *m_grp* (graphics, terminal dependent). Return value is the previous modes. Default is no change. For example, cursor($m_rev|$m_blk) sets the window cursors to blinking reverse video.

echo([window], [<string-list>])

Write the list of strings, *<string-list>*, to *window*, separated by spaces and terminated with a new line. The strings are only displayed in the window, the processes in the window are not involved (see *write* below). No value is returned. Default is the current window.

escape([escapec])

Set the escape character to *escape-char*. Returns the old escape character as a one character string. Default is no change. *Escapec* can be a string of a single character, or in the form *^X*, meaning control-*X*.

foreground([window], [flag])

Move *window* in or out of foreground. *Flag* can be one of *on*, *off*, *yes*, *no*, *true*, or *false*, with obvious meanings, or it can be a numeric expression, in which case a non-zero value is true. Returns the old foreground flag as a number. Default for *window* is the current window, default for *flag* is no change.

label([window], [label])

Set the label of *window* to *label*. Returns the old label as a string. Default for *window* is the current window, default for *label* is no change. To turn off a label, set it to an empty string ("").

list() No arguments. List the identifiers and labels of all windows. No value is returned.

nline([nline])
Set the default buffer size to *nline*. Initially, it is 48 lines. Returns the old default buffer size. Default is no change. Using a very large buffer can slow the program down considerably.

select([window])
Make *window* the current window. The previous current window is returned. Default is no change.

shell([<string-list>])
Set the default window shell program to *<string-list>*. Returns the first string in the old shell setting. Default is no change. Initially, the default shell is taken from the environment variable *SHELL*.

source(filename)
Read and execute the long commands in *filename*. Returns -1 if the file cannot be read, 0 otherwise.

terse([flag])
Set terse mode to *flag*. In terse mode, the command window stays hidden even in command mode, and errors are reported by sounding the terminal's bell. *Flag* can take on the same values as in *foreground* above. Returns the old terse flag. Default is no change.

unalias(alias)
Undefine *alias*. Returns -1 if *alias* does not exist, 0 otherwise.

unset(variable)
Undefine *variable*. Returns -1 if *variable* does not exist, 0 otherwise.

variables()
No arguments. List all variables. No value is returned.

window([row], [column], [nrow], [ncol], [nline], [frame], [pty], [mapnl], [shell])
Open a window with upper left corner at *row*, *column* and size *nrow*, *ncol*. If *nline* is specified, then that many lines are allocated for the text buffer. Otherwise, the default buffer size is used. Default values for *row*, *column*, *nrow*, and *ncol* are, respectively, the upper, left-most, lower, or right-most extremes of the screen. *Frame*, *pty*, and *mapnl* are flag values interpreted in the same way as the argument to *foreground* (see above); they mean, respectively, put a frame around this window (default true), allocate pseudo-terminal for this window rather than socketpair (default true), and map new line characters in this window to carriage return and line feed (default true if socketpair is used, false otherwise). *Shell* is a list of strings that will be used as the shell program to place in the window (default is the program specified by *shell*, see below). The created window's identifier is returned as a number.

write([window], [<string-list>])
Send the list of strings, *<string-list>*, to *window*, separated by spaces but not terminated with a new line. The strings are actually given to the window as input. No value is returned. Default is the current window.

PREDEFINED VARIABLES

These variables are for information only. Redefining them does not affect the internal operation of *window*.

baud The baud rate as a number between 50 and 38400.

modes The display modes (reverse video, underline, blinking, graphics) supported by the physical terminal. The value of *modes* is the bitwise or of some of the one bit values, *m_blk*, *m_grp*, *m_rev*, and *m_ul* (see below). These values are useful in setting the window cursors' modes (see *cursormodes* above).

m_blk The blinking mode bit.

m_grp The graphics mode bit (not very useful).

m_rev The reverse video mode bit.

m_ul The underline mode bit.

ncol The number of columns on the physical screen.

nrow The number of rows on the physical screen.

term The terminal type. The standard name, found in the second name field of the terminal's *TERMCAP* entry, is used.

FILES

~/.windowrc	startup command file.
/dev/[pt]ty[pq]?	pseudo-terminal devices.

DIAGNOSTICS

Should be self explanatory.

BUGS

NAME

write − write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message

Message from yourname@yoursystem on yourttyname at time...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a distinctive signal—**(o)** for 'over' is conventional—that the other may reply. **(oo)** for 'over and out' is suggested when conversation is about to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

SEE ALSO

mesg(1), who(1), mail(1)

NAME

xsend, xget, enroll − secret mail

SYNOPSIS

xsend person
xget
enroll

DESCRIPTION

These commands implement a secure communication channel; it 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. (However, it will accept 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

It should be integrated with ordinary mail. The announcement of secret mail makes traffic analysis possible.

NAME

xstr − extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**−c**] [**−**] [file]

DESCRIPTION

Xstr maintains a file *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

will extract 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 suffices 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.

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*.

If the **−v** flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **−d** flag is used, the file *y.tab.h* is generated 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 by S. C. Johnson.

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 rubout.

NAME

aardvark − yet another exploration game

SYNOPSIS

/usr/games/aardvark

DESCRIPTION

Aardvark is yet another computer fantasy simulation game of the adventure/zork genre. This one is written in DDL (Dungeon Definition Language) and is intended primarily as an example of how to write a dungeon in DDL.

FILES

/usr/games/lib/ddlrun ddl interpreter
/usr/games/lib/aardvarkinternal form of aardvark dungeon

AUTHOR

Mike Urban, UCLA

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-descriptive 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. After 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

SYNOPSIS

/usr/games/backgammon

DESCRIPTION

This program does what you expect. It will ask whether you need instructions.

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 great enough that you may 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.

AUTHOR

Mark Horton

NAME

battlestar − a tropical adventure game

SYNOPSIS

battlestar [**-r (recover a saved game)**]

DESCRIPTION

Battlestar is an adventure game in the classic style. However, It's slightly less of a puzzle and more a game of exploration. There are a few magical words in the game, but on the whole, simple English should suffice to make one's desires understandable to the parser.

THE SETTING

In the days before the darkness came, when battlestars ruled the heavens...

> Three He made and gave them to His daughters,
> Beautiful nymphs, the goddesses of the waters.
> One to bring good luck and simple feats of wonder,
> Two to wash the lands and churn the waves asunder,
> Three to rule the world and purge the skies with thunder.

In those times great wizards were known and their powers were beyond belief. They could take any object from thin air, and, uttering the word 'su' could disappear.

In those times men were known for their lust of gold and desire to wear fine weapons. Swords and coats of mail were fashioned that could withstand a laser blast.

But when the darkness fell, the rightful reigns were toppled. Swords and helms and heads of state went rolling across the grass. The entire fleet of battlestars was reduced to a single ship.

SAMPLE COMMANDS

```
take     ---     take an object
drop     ---     drop an object

wear     ---     wear an object you are holding
draw     ---     carry an object you are wearing

puton    ---     take an object and wear it
take off --      draw an object and drop it

throw  <object> <direction>

!        <shell esc>
```

IMPLIED OBJECTS

```
>-: take watermelon
watermelon:
Taken.
>-: eat
watermelon:
Eaten.
>-: take knife and sword and apple, drop all
knife:
Taken.
broadsword:
Taken.
apple:
```

```
Taken.
knife:
Dropped.
broadsword:
Dropped.
apple:
Dropped.
>-: get
knife:
Taken.
```

Notice that the "shadow" of the next word stays around if you want to take advantage of it. That is, saying "take knife" and then "drop" will drop the knife you just took.

SCORE & INVEN

The two commands "score" and "inven" will print out your current status in the game.

SAVING A GAME

The command "save" will save your game in a file called "Bstar." You can recover a saved game by using the "-r" option when you start up the game.

DIRECTIONS

The compass directions N, S, E, and W can be used if you have a compass. If you don't have a compass, you'll have to say R, L, A, or B, which stand for Right, Left, Ahead, and Back. Directions printed in room descriptions are always printed in R, L, A, & B relative directions.

HISTORY

I wrote Battlestar in 1979 in order to experiment with the niceties of the C Language. Most interesting things that happen in the game are hardwired into the code, so don't send me any hate mail about it! Instead, enjoy art for art's sake!

AUTHOR

David Riggle

INSPIRATION & ASSISTANCE

Chris Guthrie
Peter Da Silva
Kevin Brown
Edward Wang
Ken Arnold & Company

BUGS

Countless.

FAN MAIL

Send to edward%ucbarpa@Berkeley.arpa, chris%ucbcory@berkeley.arpa, riggle.pa@xerox.arpa.

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

boggle − play the game of boggle

SYNOPSIS

/usr/games/boggle [**+**] [**++**]

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* ``**boggle appl epie moth erhd**'') the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

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 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 one's 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 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.

AUTHORS

Originally written: Steve Levine
Further random hacking by: Steve Feldman, Kirk McKusick, Mikey Olson, and Eric Allman.

NAME

chess − the game of chess

SYNOPSIS

/usr/games/chess

DESCRIPTION

Chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

FILES

/usr/lib/chess binary image to run in compatibility mode

DIAGNOSTICS

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

BUGS

Pawns may be promoted only to queens. The book of opening moves has disappeared.

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 have the values six or nine, they are moving lines; for each 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

cribbage − the card game cribbage

SYNOPSIS

/usr/games/cribbage [**−req**] *name* ...

DESCRIPTION

Cribbage plays the card game cribbage, with the program playing one hand and the user the other. The program will initially ask the user if the rules of the game are needed − if so, it will print out the appropriate section from *According to Hoyle* with *more (I).*

Cribbage options include:

−e When the player makes a mistake scoring his hand or crib, provide an explanation of the correct score. (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, the program will randomly cut the deck.

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' will result in a short game, any other response will play a long game.

At the start of the first game, the program 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, the program 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, the program 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 latter 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. The program keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. The program 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.

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 it was desired to discard the king of diamonds, any of the following could be typed: ''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

AUTHORS

Earl T. Cohen wrote the logic. Ken Arnold added the screen oriented interface.

NAME

doctor − interact with a psychoanalyst

SYNOPSIS

/usr/games/doctor

DESCRIPTION

Doctor is a lisp-language version of the legendary ELIZA program of Joseph Weizenbaum. This script "simulates" a Rogerian psychoanalyst. Type in lower case, and when you get tired or bored, type your interrupt character (either control-C or Rubout). Remember to type two carriage returns when you want it to answer.

In order to run this you must have a Franz Lisp system in /usr/ucb/lisp.

AUTHORS

Adapted for Lisp by Jon L White, moved to Franz by John Foderaro, from an original script by Joseph Weizenbaum.

NAME

fish − play ''Go Fish''

SYNOPSIS

/usr/games/fish

DESCRIPTION

Fish plays the game of ''Go Fish'', a childrens' card game. The Object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; The default is pretty dumb.

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

AUTHOR

Ken Arnold

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

AUTHOR

Ken Arnold

NAME

hunt − a multi-player multi-terminal game

SYNOPSIS

/usr/games/hunt [-q] [**-m**] [hostname] [**-l** name]

DESCRIPTION

The object of the game *hunt* is to kill off the other players. There are no rooms, no treasures, and no monsters. Instead, you wander around a maze, find grenades, trip mines, and shoot down walls and players. The more players you kill before you die, the better your score is. If the **−m** flag is given, you enter the game as a monitor (you can see the action but you cannot play).

Hunt normally looks for an active game on the local network; if none is found, it starts one up on the local host. One may specify the location of the game by giving the *hostname* argument. The player name may be specified on the command line by using the **-l** option. This command syntax was chosen for *rlogin/rsh* compatibility. If the **−q** flag is given, *hunt* queries the network and reports if an active game were found. This is useful for .login scripts.

Hunt only works on crt (vdt) terminals with at least 24 lines, 80 columns, and cursor addressing. The screen is divided in to 3 areas. On the right hand side is the status area. It shows you how much damage you've sustained, how many charges you have left, who's in the game, who's scanning (the asterisk in front of the name), who's cloaked (the plus sign in front of the name), and other players' scores. Most of the rest of the screen is taken up by your map of the maze, except for the 24th line, which is used for longer messages that don't fit in the status area.

Hunt uses the same keys to move as *vi* does, *i.e.*, **h**,**j**,**k**, and **l** for left, down, up, right respectively. To change which direction you're facing in the maze, use the upper case version of the movement key (*i.e.*, HJKL).

Other commands are:

f – Fire (in the direction you're facing) (Takes 1 charge)
g – Throw grenade (in the direction you're facing) (Takes 9 charges)
F – Throw satchel charge (Takes 25 charges)
G – Throw bomb (Takes 49 charges)
o – Throw small slime bomb (Takes 15 charges)
O – Throw big slime bomb (Takes 30 charges)
s – Scan (show where other players are) (Takes 1 charge)
c – Cloak (hide from scanners) (Takes 1 charge)

^L – Redraw screen
q – Quit

Knowing what the symbols on the screen often helps:

−|+ – walls
/\ – diagonal (deflecting) walls
– doors (dispersion walls)
; – small mine
g – large mine
: – shot
o – grenade
O – satchel charge
@ – bomb
s – small slime bomb
$ – big slime bomb
> < ^ v – you facing right, left, up, or down
} { i ! – other players facing right, left, up, or down

```
*       – explosion
\ | /
–*–     – grenade and large mine explosion
/ | \
```

Satchel and bomb explosions are larger than grenades (5x5, 7x7, and 3x3 respectively).

Other helpful hints:

- You can only fire in the direction you are facing.
- You can only fire three shots in a row, then the gun must cool.
- A shot only affects the square it hits.
- Shots and grenades move 5 times faster than you do.
- To stab someone, you must face that player and move at them.
- Stabbing does 2 points worth of damage and shooting does 5 points.
- Slime does 5 points of damage each time it hits.
- You start with 15 charges and get 5 more for every new player.
- A grenade affects the nine squares centered about the square it hits.
- A satchel affects the twenty-five squares centered about the square it hits.
- A bomb affects the forty-nine squares centered about the square it hits.
- Slime affects all squares it oozes over (15 or 30 respectively).
- One small mine and one large mine is placed in the maze for every new player. A mine has a 5% probability of tripping when you walk directly at it; 50% when going sideways on to it; 95% when backing up on to it. Tripping a mine costs you 5 points or 10 points respectively. Defusing a mine is worth 1 charge or 9 charges respectively.
- You cannot see behind you.
- Scanning lasts for (20 times the number of players) turns. Scanning takes 1 ammo charge, so don't waste all your charges scanning.
- Cloaking lasts for 20 turns.
- Whenever you kill someone, you get 2 more damage capacity points and 2 damage points taken away.
- Maximum typeahead is 5 characters.
- A shot destroys normal (*i.e.,* non-diagonal, non-door) walls.
- Diagonal walls deflect shots and change orientation.
- Doors disperse shots in random directions (up, down, left, right).
- Diagonal walls and doors cannot be destroyed by direct shots but may be destroyed by an adjacent grenade explosion.
- Slime goes around walls, not through them.
- Walls regenerate, reappearing in the order they were destroyed. One percent of the regenerated walls will be diagonal walls or doors. When a wall is generated directly beneath a player, he is thrown in a random direction for a random period of time. When he lands, he sustains damage (up to 20 percent of the amount of damage he had before impact); that is, the less damage he had, the more nimble he is and therefore less likely to hurt himself on landing.

- The environment variable **HUNT** is checked to get the player name. If you don't have this variable set, *hunt* will ask you what name you want to play under. If it is set, you may also set up a single character keyboard map, but then you have to enumerate the options:
  e.g. setenv HUNT ''name=Sneaky,mapkey=zoFfGg1f2g3F4G''
  sets the player name to Sneaky, and the maps **z** to **o**, **F** to **f**, **G** to **g**, **1** to **f**, **2** to **g**, **3** to **F**, and **4** to **G**. The *mapkey* option must be last.
- It's a boring game if you're the only one playing.

Your score is the ratio of number of kills to number of times you entered the game and is only kept for the duration of a single session of *hunt*.

Hunt normally drives up the load average to be about (number_of_players + 0.5) greater than it would be without a *hunt* game executing. A limit of three players per host and nine players total is enforced by *hunt*.

FILES

/usr/games/lib/hunt.driver game coordinator

AUTHORS

Conrad Huang, Ken Arnold, and Greg Couch; University of California, San Francisco, Computer Graphics Lab

ACKNOWLEDGEMENTS

We thank Don Kneller, John Thomason, Eric Pettersen, and Scott Weiner for providing endless hours of play-testing to improve the character of the game. We hope their significant others will forgive them; we certainly don't.

BUGS

To keep up the pace, not everything is as realistic as possible.

There were some bugs in early releases of 4.2 BSD that *hunt* helped discover; *hunt* will crash your system if those bugs haven't been fixed.

NAME

mille − play Mille Bournes

SYNOPSIS

/usr/games/mille [file]

DESCRIPTION

Mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bournes 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 <RETURN> or <SPACE>. The <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 <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 <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.

AUTHOR

Ken Arnold
(The game itself is a product of Parker Brothers, Inc.)

SEE ALSO

curses(3X), *Screen Updating and Cursor Movement Optimization: A Library Package*, Ken Arnold

CARDS

Here is some useful information. The number in parentheses 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 this 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. 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 nastiness.

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 other than 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 Fourré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bournes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see ''Scoring'' below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.
Each Safety: 100 points for each safety in the Safety area.
All 4 Safeties: 300 points if all four safeties are played.
Each Coup Fouré: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.
Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.
Delayed Action: 300 points bonus for finishing after the deck was exhausted.
Extension: 200 points bonus for completing a 1000 mile trip.
Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop − Monopoly game

SYNOPSIS

/usr/games/monop [file]

DESCRIPTION

Monop is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property goes 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*.

AUTHOR

Ken Arnold

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

AUTHOR

Eric P. Scott

NAME

robots − fight off villainous robots

SYNOPSIS

/usr/games/robots [**−sjta**] [**scorefile**]

DESCRIPTION

Robots pits you against evil robots, who are trying to kill you (which is why they are evil). Fortunately for you, even though they are evil, they are not very bright and have a habit of bumping into each other, thus destroying themselves. In order to survive, you must get them to kill each other off, since you have no offensive weaponry.

Since you are stuck without offensive weaponry, you are endowed with one piece of defensive weaponry: a teleportation device. When two robots run into each other or a junk pile, they die. If a robot runs into you, you die. When a robot dies, you get 10 points, and when all the robots die, you start on the next field. This keeps up until they finally get you.

Robots are represented on the screen by a '**+**', the junk heaps from their collisions by a '∗', and you (the good guy) by a '@'.

The commands are:

h	move one square left
l	move one square right
k	move one square up
j	move one square down
y	move one square up and left
u	move one square up and right
b	move one square down and left
n	move one square down and right
.	(also space) do nothing for one turn
HJKLBNYU	run as far as possible in the given direction
>	do nothing for as long as possible
t	teleport to a random location
w	wait until you die or they all do
q	quit
^L	redraw the screen

All commands can be preceded by a count.

If you use the '**w**' command and survive to the next level, you will get a bonus of 10% for each robot which died after you decided to wait. If you die, however, you get nothing. For all other commands, the program will save you from typos by stopping short of being eaten. However, with '**w**' you take the risk of dying by miscalculation.

Only five scores are allowed per user on the score file. If you make it into the score file, you will be shown the list at the end of the game. If an alternate score file is specified, that will be used instead of the standard file for scores.

The options are

−s Don't play, just show the score file

−j Jump, *i.e.*, when you run, don't show any intermediate positions; only show things at the end. This is useful on slow terminals.

−t Teleport automatically when you have no other option. This is a little disconcerting until you get used to it, and then it is very nice.

−a Advance into the higher levels directly, skipping the lower, easier levels.

AUTHOR

Ken Arnold

FILES

/usr/games/lib/robots_roll   the score file

BUGS

Bugs? You *crazy*, man?!?

NAME

rogue − Exploring The Dungeons of Doom

SYNOPSIS

/usr/games/rogue [**−r**] [*save_file*] [**−s**] [**−d**]

DESCRIPTION

Rogue is a computer fantasy game with a new twist. It is crt oriented and the object of the game is to survive the attacks of various monsters and get a lot of gold, rather than the puzzle solving orientation of most computer fantasy games.

To get started you really only need to know two commands. The command **?** will give you a list of the available commands and the command **/** will identify the things you see on the screen.

To win the game (as opposed to merely playing to beat other people's high scores) you must locate the Amulet of Yendor which is somewhere below the 20th level of the dungeon and get it out. Nobody has achieved this yet and if somebody does, they will probably go down in history as a hero among heroes.

When the game ends, either by your death, when you quit, or if you (by some miracle) manage to win, *rogue* will give you a list of the top-ten scorers. The scoring is based entirely upon how much gold you get. There is a 10% penalty for getting yourself killed.

If *save_file* is specified, rogue will be restored from the specified saved game file. If the **−r** option is used, the save game file is presumed to be the default.

The **−s** option will print out the list of scores.

The **−d** option will kill you and try to add you to the score file.

For more detailed directions, read the document *A Guide to the Dungeons of Doom.*

AUTHORS

Michael C. Toy, Kenneth C. R. C. Arnold, Glenn Wichman

FILES

/usr/games/lib/rogue_roll	Score file
~/rogue.save	Default save file

SEE ALSO

Michael C. Toy and Kenneth C. R. C. Arnold, *A guide to the Dungeons of Doom*

BUGS

Probably infinite (although countably infinite). However, that Ice Monsters sometimes transfix you permanently is *not* a bug. It's a feature.

NAME

sail − multi-user wooden ships and iron men

SYNOPSIS

sail [**−s** [**−l**]] [**−x**] [**−b**] [**num**]

DESCRIPTION

Sail is a computer version of Avalon Hill's game of fighting sail originally developed by S. Craig Taylor.

Players of *Sail* take command of an old fashioned Man of War and fight other players or the computer. They may re-enact one of the many historical sea battles recorded in the game, or they can choose a fictional battle.

As a sea captain in the *Sail* Navy, the player has complete control over the workings of his ship. He must order every maneuver, change the set of his sails, and judge the right moment to let loose the terrible destruction of his broadsides. In addition to fighting the enemy, he must harness the powers of the wind and sea to make them work for him. The outcome of many battles during the age of sail was decided by the ability of one captain to hold the 'weather gage.'

The flags are:

−s Print the names and ships of the top ten sailors.

−l Show the login name. Only effective with **-s**.

−x Play the first available ship instead of prompting for a choice.

−b No bells.

IMPLEMENTATION

Sail is really two programs in one. Each player starts up a process which runs his own ship. In addition, a *driver* process is forked (by the first player) to run the computer ships and take care of global bookkeeping.

Because the *driver* must calculate moves for each ship it controls, the more ships the computer is playing, the slower the game will appear.

If a player joins a game in progress, he will synchronize with the other players (a rather slow process for everyone), and then he may play along with the rest.

To implement a multi-user game in Version 7 UNIX, which was the operating system *Sail* was first written under, the communicating processes must use a common temporary file as a place to read and write messages. In addition, a locking mechanism must be provided to ensure exclusive access to the shared file. For example, *Sail* uses a temporary file named /tmp/#sailsink.21 for scenario 21, and corresponding file names for the other scenarios. To provide exclusive access to the temporary file, *Sail* uses a technique stolen from an old game called "pubcaves" by Jeff Cohen. Processes do a busy wait in the loop

```
for (n = 0; link(sync_file, sync_lock) < 0 && n < 30; n++)
        sleep(2);
```

until they are able to create a link to a file named "/tmp/#saillock.??". The "??" correspond to the scenario number of the game. Since UNIX guarantees that a link will point to only one file, the process that succeeds in linking will have exclusive access to the temporary file.

Whether or not this really works is open to speculation. When ucbmiro was rebooted after a crash, the file system check program found 3 links between the *Sail* temporary file and its link file.

CONSEQUENCES OF SEPARATE PLAYER AND DRIVER

When players do something of global interest, such as moving or firing, the driver must coordinate the action with the other ships in the game. For example, if a player wants to move in a certain direction, he writes a message into the temporary file requesting the driver to move his ship. Each ''turn,'' the driver reads all the messages sent from the players and decides what happened. It then writes back into the temporary file new values of variables, etc.

The most noticeable effect this communication has on the game is the delay in moving. Suppose a player types a move for his ship and hits return. What happens then? The player process saves up messages to be written to the temporary file in a buffer. Every 7 seconds or so, the player process gets exclusive access to the temporary file and writes out its buffer to the file. The driver, running asynchronously, must read in the movement command, process it, and write out the results. This takes two exclusive accesses to the temporary file. Finally, when the player process gets around to doing another 7 second update, the results of the move are displayed on the screen. Hence, every movement requires four exclusive accesses to the temporary file (anywhere from 7 to 21 seconds depending upon asynchrony) before the player sees the results of his moves.

In practice, the delays are not as annoying as they would appear. There is room for "pipelining" in the movement. After the player writes out a first movement message, a second movement command can then be issued. The first message will be in the temporary file waiting for the driver, and the second will be in the file buffer waiting to be written to the file. Thus, by always typing moves a turn ahead of the time, the player can sail around quite quickly.

If the player types several movement commands between two 7 second updates, only the last movement command typed will be seen by the driver. Movement commands within the same update "overwrite" each other, in a sense.

THE HISTORY OF SAIL

I wrote the first version of *Sail* on a PDP 11/70 in the fall of 1980. Needless to say, the code was horrendous, not portable in any sense of the word, and didn't work. The program was not very modular and had fseeks() and fwrites() every few lines. After a tremendous rewrite from the top down, I got the first working version up by 1981. There were several annoying bugs concerning firing broadsides and finding angles. *Sail* uses no floating point, by the way, so the direction routines are rather tricky. Ed Wang rewrote my angle() routine in 1981 to be more correct (although it still doesn't work perfectly), and he added code to let a player select which ship he wanted at the start of the game (instead of the first one available).

Captain Happy (Craig Leres) is responsible for making *Sail* portable for the first time. This was no easy task, by the way. Constants like 2 and 10 were very frequent in the code. I also became famous for using "Riggle Memorial Structures" in *Sail.* Many of my structure references are so long that they run off the line printer page. Here is an example, if you promise not to laugh.

specs[scene[flog.fgamenum].ship[flog.fshipnum].shipnum].pts

Sail received its fourth and most thorough rewrite in the summer and fall of 1983. Ed Wang rewrote and modularized the code (a monumental feat) almost from scratch. Although he introduced many new bugs, the final result was very much cleaner and (?) faster. He added window movement commands and find ship commands.

HISTORICAL INFO

Old Square Riggers were very maneuverable ships capable of intricate sailing. Their only disadvantage was an inability to sail very close to the wind. The design of a 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 was small compared to a 68 gun broadside of 24 or 32 pounders. The guns bear approximately like so:

```
   \
    b----------------
 ---0
    \
     \
```

```
\    up to a range of ten (for round shot)
 \
  \
   \
```

An interesting phenomenon occurred when a broadside was fired 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. Because the bows of a ship are very strong and present a smaller target than the stern, a stern rake (firing from the stern to the bow) causes more damage than a bow rake.

```
 b
00   ----  Stern rake!
  a
```

Most ships were equipped with carronades, which were very large, close range cannons. American ships from the revolution until the War of 1812 were almost entirely armed with carronades.

The period of history covered in *Sail* is approximately from the 1770's until the end of Napoleanic France in 1815. There are many excellent books about the age of sail. My favorite author is Captain Frederick Marryat. More contemporary authors include C.S. Forester and Alexander Kent.

Fighting ships came in several sizes classed by armament. The mainstays of any fleet were its "Ships of the Line", or "Line of Battle Ships". They were so named because these ships fought together in great lines. They were close enough for mutual support, yet every ship could fire both its broadsides. We get the modern words "ocean liner," or "liner," and "battleship" from "ship of the line." The most common size was the the 74 gun two decked ship of the line. The two gun decks usually mounted 18 and 24 pounder guns.

The pride of the fleet were the first rates. These were huge three decked ships of the line mounting 80 to 136 guns. The guns in the three tiers were usually 18, 24, and 32 pounders in that order from top to bottom.

Various other ships came next. They were almost all "razees," or ships of the line with one deck sawed off. They mounted 40-64 guns and were a poor cross between a frigate and a line of battle ship. They neither had the speed of the former nor the firepower of the latter.

Next came the "eyes of the fleet." Frigates came in many sizes mounting anywhere from 32 to 44 guns. They were very handy vessels. They could outsail anything bigger and outshoot anything smaller. Frigates didn't fight in lines of battle as the much bigger 74's did. Instead, they harassed the enemy's rear or captured crippled ships. They were much more useful in missions away from the fleet, such as cutting out expeditions or boat actions. They could hit hard and get away fast.

Lastly, there were the corvettes, sloops, and brigs. These were smaller ships mounting typically fewer than 20 guns. A corvette was only slightly smaller than a frigate, so one might have up to 30 guns. Sloops were used for carrying dispatches or passengers. Brigs were something you built for land-locked lakes.

SAIL PARTICULARS

Ships in *Sail* are represented by two characters. One character represents the bow of the ship, and the other represents the stern. Ships have nationalities and numbers. The first ship of a nationality is number 0, the second number 1, etc. Therefore, the first British ship in a game would be printed as "b0". The second Brit would be "b1", and the fifth Don would be "s4".

Ships can set normal sails, called Battle Sails, or bend on extra canvas called Full Sails. A ship under full sail is a beautiful sight indeed, and it can move much faster than a ship under Battle Sails. The only trouble is, with full sails set, there is so much tension on sail and rigging that a well aimed round shot can burst a sail into ribbons where it would only cause a little hole in a loose sail. For this reason, rigging damage is doubled on a ship with full sails set. Don't let that discourage you from using full sails. I like to keep them up right into the heat of battle. A ship with full sails set has a capital letter for its nationality. E.g., a Frog, "f0", with full sails set would be printed as "F0".

When a ship is battered into a listing hulk, the last man aboard "strikes the colors." This ceremony is the ship's formal surrender. The nationality character of a surrendered ship is printed as "!". E.g., the Frog of our last example would soon be "!0".

A ship has a random chance of catching fire or sinking when it reaches the stage of listing hulk. A sinking ship has a "~" printed for its nationality, and a ship on fire and about to explode has a "#" printed.

Captured ships become the nationality of the prize crew. Therefore, if an American ship captures a British ship, the British ship will have an "a" printed for its nationality. In addition, the ship number is changed to "&","’", "(", ,")", "*", or "+" depending upon the original number, be it 0,1,2,3,4, or 5. E.g., the "b0" captured by an American becomes the "a&". The "s4" captured by a Frog becomes the "f*".

The ultimate example is, of course, an exploding Brit captured by an American: "#&".

MOVEMENT

Movement is the most confusing part of *Sail* to many. Ships can head in 8 directions:

```
                        0       0       0
b       b       b0      b       b       b       0b      b
0       0                                               0
```

The stern of a ship moves when it turns. The bow remains stationary. Ships can always turn, regardless of the wind (unless they are becalmed). All ships drift when they lose headway. If a ship doesn't move forward at all for two turns, it will begin to drift. If a ship has begun to drift, then it must move forward before it turns, if it plans to do more than make a right or left turn, which is always possible.

Movement commands to *Sail* are a string of forward moves and turns. An example is "l3". It will turn a ship left and then move it ahead 3 spaces. In the drawing above, the "b0" made 7 successive left turns. When *Sail* prompts you for a move, it prints three characters of import. E.g.,

move (7, 4):

The first number is the maximum number of moves you can make, including turns. The second number is the maximum number of turns you can make. Between the numbers is sometimes printed a quote "’". If the quote is present, it means that your ship has been drifting, and you must move ahead to regain headway before you turn (see note above). Some of the possible moves for the example above are as follows:

```
move (7, 4): 7
move (7, 4): 1
move (7, 4): d            /* drift, or do nothing */
move (7, 4): 6r
move (7, 4): 5r1
move (7, 4): 4r1r
move (7, 4): l1r1r2
move (7, 4): 1r1r1r1
```

Because square riggers performed so poorly sailing into the wind, if at any point in a movement command you turn into the wind, the movement stops there. E.g.,

```
move (7, 4): ll4
Movement Error;
Helm: ll
```

Moreover, whenever you make a turn, your movement allowance drops to min(what's left, what you would have at the new attitude). In short, if you turn closer to the wind, you most likely won't be able to sail the full allowance printed in the "move" prompt.

Old sailing captains had to keep an eye constantly on the wind. Captains in *Sail* are no different. A ship's ability to move depends on its attitide to the wind. The best angle possible is to have the wind off your quarter, that is, just off the stern. The direction rose on the side of the screen gives the possible movements for your ship at all positions to the wind. Battle sail speeds are given first, and full sail speeds are given in parenthesis.

```
 0 1(2)
\|/
-^-3(6)
/|\
 | 4(7)
3(6)
```

Pretend the bow of your ship (the "^") is pointing upward and the wind is blowing from the bottom to the top of the page. The numbers at the bottom "3(6)" will be your speed under battle or full sails in such a situation. If the wind is off your quarter, then you can move "4(7)". If the wind is off your beam, "3(6)". If the wind is off your bow, then you can only move "1(2)". Facing into the wind, you can't move at all. Ships facing into the wind were said to be "in irons".

WINDSPEED AND DIRECTION

The windspeed and direction is displayed as a little weather vane on the side of the screen. The number in the middle of the vane indicates the wind speed, and the + to - indicates the wind direction. The wind blows from the + sign (high pressure) to the - sign (low pressure). E.g.,

```
|
3
+
```

The wind speeds are 0 = becalmed, 1 = light breeze, 2 = moderate breeze, 3 = fresh breeze, 4 = strong breeze, 5 = gale, 6 = full gale, 7 = hurricane. If a hurricane shows up, all ships are destroyed.

GRAPPLING AND FOULING

If two ships collide, they run the risk of becoming tangled together. This is called "fouling." Fouled ships are stuck together, and neither can move. They can unfoul each other if they want to. Boarding parties can only be sent across to ships when the antagonists are either fouled or grappled.

Ships can grapple each other by throwing grapnels into the rigging of the other.

The number of fouls and grapples you have are displayed on the upper right of the screen.

BOARDING

Boarding was a very costly venture in terms of human life. Boarding parties may be formed in *Sail* to either board an enemy ship or to defend your own ship against attack. Men organized as Defensive Boarding Parties fight twice as hard to save their ship as men left unorganized.

The boarding strength of a crew depends upon its quality and upon the number of men sent.

CREW QUALITY

The British seaman was world renowned for his sailing abilities. American sailors, however, were actually the best seamen in the world. Because the American Navy offered twice the wages of the Royal Navy, British seamen who liked the sea defected to America by the thousands.

In *Sail,* crew quality is quantized into 5 energy levels. "Elite" crews can outshoot and outfight all other sailors. "Crack" crews are next. "Mundane" crews are average, and "Green" and "Mutinous" crews are below average. A good rule of thumb is that "Crack" or "Elite" crews get one extra hit per broadside compared to "Mundane" crews. Don't expect too much from "Green" crews.

BROADSIDES

Your two broadsides may be loaded with four kinds of shot: grape, chain, round, and double. You have guns and carronades in both the port and starboard batteries. Carronades only have a range of two, so you have to get in close to be able to fire them. You have the choice of firing at the hull or rigging of another ship. If the range of the ship is greater than 6, then you may only shoot at the rigging.

The types of shot and their advantages are:

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. Sometimes devastating against enemy crews.

On the side of the screen is displayed some vital information about your ship:

```
Load  D! R!
Hull  9
Crew  4  4  2
Guns  4  4
Carr  2  2
Rigg  5 5 5 5
```

"Load" shows what your port (left) and starboard (right) broadsides are loaded with. A "!" after the type of shot indicates that it is an initial broadside. Initial broadside were loaded with care before battle and before the decks ran red with blood. As a consequence, initial broadsides are a little more effective than broadsides loaded later. A "*" after the type of shot indicates that the gun crews are still loading it, and you cannot fire yet. "Hull" shows how much hull you have left. "Crew" shows your three sections of crew. As your crew dies off, your ability to fire decreases. "Guns" and "Carr" show your port and starboard guns. As you lose guns, your ability to fire decreases. "Rigg" shows how much rigging you have on your 3 or 4 masts. As rigging is shot away, you lose mobility.

EFFECTIVENESS OF FIRE

It is very dramatic when a ship fires its thunderous broadsides, but the mere opportunity to fire them does not guarantee any hits. Many factors influence the destructive force of a broadside. First of all, and the chief factor, is distance. It is harder to hit a ship at range ten than it is to hit one sloshing alongside. Next is raking. Raking fire, as mentioned before, can sometimes dismast a ship at range ten. Next, crew size and quality affects the damage done by a broadside. The number of guns firing also bears on the point, so to speak. Lastly, weather affects the accuracy of a broadside. If the seas are high (5 or 6), then the lower gunports of ships of the line can't even be opened to run out the guns. This gives frigates and other flush decked vessels an advantage in a storm. The scenario *Pellew vs. The Droits de L'Homme* takes advantage of this peculiar circumstance.

REPAIRS

Repairs may be made to your Hull, Guns, and Rigging at the slow rate of two points per three turns. The message "Repairs Completed" will be printed if no more repairs can be made.

PECULIARITIES OF COMPUTER SHIPS

Computer ships in *Sail* follow all the rules above with a few exceptions. Computer ships never repair damage. If they did, the players could never beat them. They play well enough as it is. As a consolation, the computer ships can fire double shot every turn. That fluke is a good reason to keep your distance. The *Driver* figures out the moves of the computer ships. It computes them with a typical A.I. distance function and a depth first search to find the maximum "score." It seems to work fairly well, although I'll be the first to admit it isn't perfect.

HOW TO PLAY

Commands are given to *Sail* by typing a single character. You will then be prompted for further input. A brief summary of the commands follows.

COMMAND SUMMARY

'f' Fire broadsides if they bear
'l' Reload
'L' Unload broadsides (to change ammo)
'm' Move
'i' Print the closest ship
'I' Print all ships
'F' Find a particular ship or ships (e.g. "a?" for all Americans)
's' Send a message around the fleet
'b' Attempt to board an enemy ship
'B' Recall boarding parties
'c' Change set of sail
'r' Repair
'u' Attempt to unfoul
'g' Grapple/ungrapple
'v' Print version number of game
'^L' Redraw screen
'Q' Quit

'C' Center your ship in the window
'U' Move window up
'D','N' Move window down
'H' Move window left
'J' Move window right
'S' Toggle window to follow your ship or stay where it is

SCENARIOS

Here is a summary of the scenarios in *Sail:*

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 the others 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 Richard Basehart and 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)

(j) Akagi 112 gun 3 Decker SOL (green crew) (27 pts)
(j) Kaga 96 gun 3 Decker SOL (green crew) (24 pts)
(j) Soryu 80 gun Ship of the Line (green crew) (23 pts)

Star Trek:

Wind from the S, blowing a fresh breeze.

(a) Enterprise 450 gun Ship of the Line (elite crew) (75 pts)
(a) Yorktown 450 gun Ship of the Line (elite crew) (75 pts)
(a) Reliant 450 gun Ship of the Line (elite crew) (75 pts)
(a) Galileo 450 gun Ship of the Line (elite crew) (75 pts)
(k) Kobayashi Maru 450 gun Ship of the Line (elite crew) (75 pts)
(k) Klingon II 450 gun Ship of the Line (elite crew) (75 pts)
(o) Red Orion 450 gun Ship of the Line (elite crew) (75 pts)
(o) Blue Orion 450 gun Ship of the Line (elite crew) (75 pts)

CONCLUSION

Sail has been a group effort.

Ken Arnold Code

curses library (pu!)

AUTHOR

Dave Riggle

CO-AUTHOR

Ed Wang

REFITTING

Craig Leres

CONSULTANTS

Chris Guthrie
Captain Happy
Horatio Nelson
Nancy Reagan
  and many valiant others...

REFERENCES

Wooden Ships & Iron Men, by Avalon Hill
Captain Horatio Hornblower Novels, (13 of them) by C.S. Forester
Captain Richard Bolitho Novels, (12 of them) by Alexander Kent
The Complete Works of Captain Frederick Marryat, (about 20) especially
  Mr. Midshipman Easy
  Peter Simple
  Jacob Faithful
  Japhet in Search of a Father
  Snarleyyow, or The Dog Fiend
  Frank Mildmay, or The Naval Officer

SEE ALSO

midway(PUBLIC)

BUGS

Probably a few, and please report them to "riggle@ernie" and "edward@arpa."

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
/usr/games/busy	program to determine if system too busy

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.

AUTHOR

Eric Allman

SEE ALSO

/usr/doc/trek

COMMAND SUMMARY

abandon	**ca**pture
cloak **u**p/**d**own	
computer request; ...	**da**mages
destruct	**do**ck
help	**i**mpulse course distance
lrscan	**m**ove course distance
phasers **a**utomatic amount	
phasers **m**anual amt1 course1 spread1 ...	
torpedo course [**y**es] angle/**n**o	
ram course distance	**r**est time
shell	**sh**ields **u**p/**d**own
srscan [**y**es/**n**o]	
status	**terminate y**es/**n**o
undock	**v**isual course
warp warp_factor	

NAME

worm − Play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In *worm,* you are a little worm, your body is the "o"'s on the screen and your head is the "@". You move with the hjkl keys (as in the game snake). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit, if your worm eats the digit is will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms − animate worms on a display terminal

SYNOPSIS

/usr/games/worms [**−field**] [**−length** #] [**−number** #] [**−trail**]

DESCRIPTION

Brian Horn (cithep!bdh) showed me a *TOPS-20* program on the DEC-2136 machine called *WORM*, and suggested that I write a similar program that would run under *Unix*. I did, and no apologies.

−field makes a "field" for the worm(s) to eat; **−trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

AUTHOR

Eric P. Scott

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

zork − the game of dungeon

SYNOPSIS

/usr/games/zork

DESCRIPTION

Dungeon is a computer fantasy simulation based on Adventure and on Dungeons & Dragons, originally written by Lebling, Blank, and Anderson of MIT. In it you explore a dungeon made up of various rooms, caves, rivers, and so on. The object of the game is to collect as much treasure as possible and stow it safely in the trophy case (and, of course, to stay alive.)

Figuring out the rules is part of the game, but if you are stuck, you should start off with ''open mailbox'', ''take leaflet'', and then ''read leaflet''. Additional useful commands that are not documented include:

quit	(to end the game)
!cmd	(the usual shell escape convention)
>	(to save a game)
<	(to restore a game)

FILES

/usr/games/lib/d*

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
environ	user environment
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
term	conventional names for terminals

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

environ − user environment

SYNOPSIS

extern char **environ;

DESCRIPTION

An array of strings called the 'environment' is made available by *execve*(2) when a process begins. By convention these strings have the form '*name=value*'. The following names are used by various commands:

PATH The sequence of directory prefixes that *sh, time, nice*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by ':'. *Login*(1) sets PATH=:/usr/ucb:/bin:/usr/bin.

HOME A user's login directory, set by *login*(1) from the password file *passwd*(5).

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as *nroff* or *plot*(1G), which may exploit special terminal capabilities. See */etc/termcap* (*termcap*(5)) for a list of terminal types.

SHELL The file name of the users login shell.

TERMCAP The string describing the terminal in TERM, or the name of the termcap file, see *termcap*(5),*termcap*(3X).

EXINIT A startup list of commands read by *ex*(1), *edit*(1), and *vi*(1).

USER The login name of the user.

PRINTER The name of the default printer to be used by *lpr*(1), *lpq*(1), and *lprm*(1).

Further names may be placed in the environment by the *export* command and 'name=value' arguments in *sh*(1), or by the *setenv* command if you use *csh*(1). Arguments may also be placed in the environment at the point of an *execve*(2). It is unwise to conflict with certain *sh*(1) variables that are frequently exported by '.profile' files: MAIL, PS1, PS2, IFS.

SEE ALSO

csh(1), ex(1), login(1), sh(1), execve(2), system(3), termcap(3X), termcap(5)

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 (4)
  MAKEDEV
    shell script to create special files
  MAKEDEV.local
    site specific part of MAKEDEV
  console main console, *tty*(4)
  tty* terminals, *tty*(4)
  hp* disks, *hp*(4)
  rhp* raw disks, *hp*(4)
  up* UNIBUS disks *up*(4)
  ...

/bin/ utility programs, cf /usr/bin/ (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. (2,3,3S)
  ...
  ccom C compiler proper
  cpp C preprocessor
  c2 C code improver
  ...

/etc/ essential data and maintenance utilities; sect (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)
  disktab disk characteristics and partition tables, *disktab*(5)
  hosts host name to network address mapping file, *hosts*(5)
  networks
    network name to network number mapping file, *networks*(5)
  protocols
    protocol name to protocol number mapping file, *protocols*(5)
  services network services definition file, *services*(5)
  remote names and description of remote hosts for *tip*(1C), *remote*(5)
  phones private phone numbers for remote hosts, as described in *phones*(5)

```
          ttys      properties of terminals, ttys(5)
          getty     part of login, getty(8)
          init      the parent of all processes, init(8)
          rc        shell program to bring the system up
          rc.local  site dependent portion of rc
          cron      the clock daemon, cron(8)
          mount     mount(8)
          ...
/sys/     system source
          h/        header (include) files
                    acct.h    acct(5)
                    stat.h    stat(2)
                    ...
          sys/      machine independent system source
                    init_main.c
                    uipc_socket.c
                    ufs_syscalls.c
                    ...
          conf/     site configuration files
                    GENERIC
                    ...
          net/      general network source
          netinet/  DARPA Internet network source
          netimp/   network code related to use of an IMP
                    if_imp.c
                    if_imphost.c
                    if_imphost.h
                    ...
          vax/      source specific to the VAX
                    locore.s
                    machdep.c
                    ...
          vaxuba/   device drivers for hardware which resides on the UNIBUS
                    uba.c
                    dh.c
                    up.c
                    ...
          vaxmba/
                    device drivers for hardware which resides on the MASBUS
                    mba.c
                    hp.c
                    ht.c
                    ...
          vaxif     network interface drivers for the VAX
                    if_en.c
                    if_ec.c
                    if_vv.c
                    ...
/tmp/     temporary files, usually on a fast device, 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		
		tracct	phototypesetter accounting, *troff*(1)		
		lpacct	line printer accounting *lpr*(1)		
		vaacct, vpacct			
			varian and versatec accounting *vpr*(1), *vtroff*(1), *pac*(8)		
/usr	/bin				
	utility programs, to keep /bin/ small				
	tmp/	temporaries, to keep /tmp/ small			
		stm*	used by *sort*(1)		
		raster	used by *plot*(1G)		
	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, *intro*(3S)		
		math.h	(3M)		
		...			
		sys/	system-defined layouts, cf /sys/h		
		net/	symbolic link to sys/net		
		machine/			
			symbolic link to sys/machine		
		...			
	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/
        volume 1 of this manual, man(1)
              man0/     general
                        intro    introduction to volume 1, ms(7) format
                        xx       template for manual page
              man1/     chapter 1
                        as.1
                        mount.1m
                        ...
              ...
              cat1/     preformatted pages for section 1
              ...
        msgs/ messages, cf msgs(1)
              bounds    highest and lowest message
        new/  binaries of new versions of programs
        preserve/
              editor temporaries preserved here after crashes/hangups
        public/ binaries of user programs - write permission to everyone
        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/
              uucp/     work files and staging area for uucp(1C)
                        LOGFILE
                                 summary log
                        LOG.*    log file for one transaction
                        mqueue/
                                 mail queue for sendmail(8)
        wd    initial working directory of a user, typically wd is the user's login name
              .profile  set environment for sh(1), environ(7)
              .project  what you are doing (used by ( finger(1) )
```

```
                    .cshrc    startup file for csh(1)
                    .exrc     startup file for ex(1)
                    .plan     what your short-term plans are (used by finger(1) )
                    .netrc    startup file for various network programs
                    .msgsrc   startup file for msgs(1)
                    .mailrc   startup file for mail(1)
                    calendar
                              user's datebook for calendar(1)
          doc/      papers, mostly in volume 2 of this manual, typically in ms(7) format
                    as/       assembler manual
                    c         C manual
                    ...
/usr/     src/
          source programs for utilities, etc.
          bin/      source of commands in /bin
                    as/       assembler
                    ar.c      source for ar(1)
                    ...
          usr.bin/  source for commands in /usr/bin
                    troff/    source for nroff and troff(1)
                              font/     source for font tables, /usr/lib/font/
                                        ftR.c     Roman
                                        ...
                              term/     terminal characteristics tables, /usr/lib/term/
                                        tab300.c
                                                  DASI 300
                                        ...
                              ...
          ucb       source for programs in /usr/ucb
          games/    source for /usr/games
          lib/      source for programs and archives in /lib
                    libc/     C runtime library
                              csu/      startup and wrapup routines needed with every C program
                                        crt0.s    regular startup
                                        mcrt0.s   modified startup for cc -p
                              sys/      system calls (2)
                                        access.s
                                        brk.s
                                        ...
                    stdio/    standard I/O functions (3S)
                              fgets.c
                              fopen.c
                              ...
                    gen/      other functions in (3)
                              abs.c
                              ...
                    net/      network functions in (3N)
                              gethostbyname.c
                              ...
          local/    source which isn't normally distributed
          new/      source for new versions of commands and library routines
          old/      source for old versions of commands and library routines
```

ucb/	binaries of programs developed at UCB	
	...	
	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), apropos(1), whatis(1), whereis(1), finger(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 protocol 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 were 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. 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 combination 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 DARPA is converting to real domains. The following rules may be useful:

• The syntax ''user@host.ARPA'' is being split up into ''user@host.COM'', ''user@host.GOV'', and ''user@host.EDU'' for commercial, government, and educational institutions respectively.

• The syntax ''user@host'' (with no dots) has traditionally referred to the ARPANET. In the future this semantic will not be continued — instead, the host will be assumed to be in your organization. You should start using one of the syntaxes above.

• Host names of the form ''ORG-NAME'' (e.g., MIT-MC or CMU-CS-A) will be changing to ''NAME.ORG.XXX'' (where 'XXX' is COM, GOV, or EDU). For example, MIT-MC will change to MC.MIT.EDU. In some cases names will be split apart even if they do not have dashes. For example, USC-ISIF will probably change to F.ISI.USC.EDU.

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. Addresses which show 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.

Other Networks.

Some other networks can be reached by giving the name of the network as the last component of the domain. *This is not a standard feature* and may not be supported at all sites. For example, messages to CSNET or BITNET sites can often be sent to ''user@host.CSNET'' or ''user@host.BITNET'' respectively.

BERKELEY

The following comments apply only to the Berkeley environment.

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!yourname%monet

Computer Center.

The Berkeley Computer Center is in a subdomain of Berkeley. Messages to the computer center should be addressed to:

user%host.CC@Berkeley.ARPA

The alternate syntax:

user@host.CC

may be used 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.

BUGS

The RFC822 group syntax (''group:user1,user2,user3;'') is not supported except in the special case of ''group:;'' because of a conflict with old berknet-style addresses.

Route-Address syntax is grotty.

UUCP- and ARPANET-style addresses do not coexist politely.

SEE ALSO

mail(1), sendmail(8); Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages,* RFC822.

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, 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 may be 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.
.GE	-	yes	End *gremlin* picture.
.GS	-	yes	Begin *gremlin* picture.
.PE	-	yes	End *pic* picture.
.PS	-	yes	Begin *pic* picture.
.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)
.bu	-	yes	Begin bulleted paragraph
.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.
.sm *x*	-	no	Set *x* in a smaller pointsize.
.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. 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, eqn,* and *refer*(1) preprocessors for equations, tables, and references is acceptable as input.

FILES

/usr/lib/tmac/tmac.x
/usr/lib/ms/x.???

SEE ALSO

eqn(1), refer(1), tbl(1), troff(1)

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	−	y	begin abstract; if *x*=no don't label abstract
.AE	−	y	end abstract
.AI	−	y	author's institution
.AM	−	n	better accent mark definitions
.AU	−	y	author's name
.B *x*	−	n	embolden *x*; if no *x*, switch to boldface
.B1	−	y	begin text to be enclosed in a box
.B2	−	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	−	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	−	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	−	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I,L,C,B; *y*=indent
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.LD	−	y	left display with no keep
.CD	−	y	centered display with no keep
.BD	−	y	block display; center entire block
.EF *x*	−	n	even page footer *x* (3 part as for .tl)
.EH *x*	−	n	even page header *x* (3 part as for .tl)
.EN	−	y	end displayed equation produced by *eqn*

.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for .tl)
.OH *x*	–	n	odd page header *x* (3 part as for .tl)
.P1	if TM	n	print header on 1st page
.PP	–	y,y	paragraph with first line indented
.PT	- % -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on 1st page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set tabs to 8n 16n ... (nroff) 5n 10n ... (troff)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by *tbl*
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in *troff*
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; *y*=indent
.XE	–	y	end index entry (or series of .IX entries)
.XP	–	y,y	paragraph with first line exdented, others indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.]-	–	n	beginning of *refer* reference
.[0	–	n	end of unclassifiable type of reference
.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in −ms by means of built-in number registers. For example, this sets the line length to 6.5 inches:

.nr LL 6.5i

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in −ms; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in *nroff,* ‘‘ in *troff*)
*U	unquote (" in *nroff,* ’’ in *troff*)
*−	dash (-- in *nroff,* — in *troff*)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*´	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

term − conventional names for terminals

DESCRIPTION

Certain commands use these terminal names. They are maintained as part of the shell environment (see *sh*(1),*environron*(7)).

adm3a	Lear Seigler Adm-3a
2621	Hewlett-Packard HP262? series terminals
hp	Hewlett-Packard HP264? series terminals
c100	Human Designed Systems Concept 100
h19	Heathkit H19
mime	Microterm mime in enhanced ACT IV mode
1620	DIABLO 1620 (and others using HyType II)
300	DASI/DTC/GSI 300 (and others using HyType I)
33	TELETYPE® Model 33
37	TELETYPE Model 37
43	TELETYPE Model 43
735	Texas Instruments TI735 (and TI725)
745	Texas Instruments TI745
dumb	terminals with no special features
dialup	a terminal on a phone line with no known characteristics
network	a terminal on a network connection with no known characteristics
4014	Tektronix 4014
vt52	Digital Equipment Corp. VT52

The list goes on and on. Consult /etc/termcap (see *termcap*(5)) for an up-to-date and locally correct list.

Commands whose behavior may depend on the terminal either consult TERM in the environment, or accept arguments of the form **−Tterm,** where *term* is one of the names given above.

SEE ALSO

stty(1), tabs(1), plot(1G), sh(1), environ(7) ex(1), clear(1), more(1), ul(1), tset(1), termcap(5), termcap(3X), ttytype(5)
troff(1) for *nroff*

BUGS

The programs that ought to adhere to this nomenclature do so only fitfully.

Colophon to the PostScript reprint

This reprint of the URM book is my first installment in the project seeking to create PostScript reprints of all 6 books of the original 4.3BSD manual set (see the introduction at the front of this book). The goal of the project is to reprint the original 4.3BSD books without any content changes (i.e., without Quasijarus changes and additions), but use the 4.3BSD-Quasijarus *pstroff* suite to accomplish the rendering of the original troff sources into PostScript.

My reference for this work, i.e., the original whose appearance I sought to match as closely as reasonably possible, is a historic print of the manual set from USENIX; the verso of the front cover reads:

4.3 BSD edition:	
First Printing	November 1986
Second Printing	June 1987
Third Printing	June 1988

Based on the appearance of the troff fonts in this historic print and on my general knowledge of UCB history, I can say with high confidence that these books were not printed via PostScript; instead they were most likely troff'ed on some non-PostScript typesetter and then mass-reproduced by analogue means. To the best of my research, the original troffing of the master delivered to USENIX was done at UCB on an Autologic APS-5 typesetter, a CRT machine similar to the Linotron 202 which had prompted the original development of *ditroff* back at Bell Labs.

1. Troff-to-PostScript issues

1.1. Research on the document sources and the original production method

Even though PostScript was already available in a number of printers and typesetters at the time of 4.3BSD, UCB did not embrace it. The *CSRG Archives* CD set distributed by Marshall Kirk McKusick includes a snapshot of */usr/src* at the time of the disbanding of the CSRG, and some of the bits under */usr/src/local* date back to the late 1980s, the time of interest for this work. The existence of */usr/src/local/transcript* with dates in that timeframe proves that they were aware of PostScript, probably in connection with having to make use of an occasional LaserWriter or two, but it certainly wasn't anywhere near the front and center of their document production processes.

The bits preserved under */usr/src/local/ditroff* are more relevant, and the bits under */usr/src/local/ditroff/ditroff.old.** appear to be the primary *ditroff* version in use internally at the time of 4.3BSD. That *ditroff* suite targeted neither PostScript nor the ''traditional'' *ditroff* typesetters like APS-5 or Linotron, instead it targeted a few select raster output devices, i.e., electrophotographic printers and plotters of that era. Their primary target has been the ''dp'' device, which appears to have been a Xerox INTERPRESS EP printer. They also had an APS-5, apparently, but it appears that they used a different version of *ditroff* for it, specifically designated as ''old'': see the two Makefiles under */usr/src/admin/admin/4.3* for a clue as to how they troffed documents for the ''dp'' versus the APS.

The document sources shipped on BSD tapes under */usr/doc* and */usr/man* are largely devoid of Makefiles, hence the troffing must have been done either by hand or by some ''secret'' scripts maintained outside of the source tree. Where Makefiles do exist, they refer to the *ditroff* toolchain for the ''dp'' device. So were the 4.3BSD book masters troffed on that ''dp'' device and not on the APS-5 as generally claimed? Well, we could have tested this hypothesis by examining the fonts used with the latter and comparing them to the appearance of the USENIX print, but unfortunately the fonts for the Xerox setup at UCB seem to have been lost. Hence we can't disprove that possibility. However, the following sources back the claim of the troffing having been done on the APS-5:

- The troff font sample in the USD:24 document is labeled ‘‘This font sample is printed on an APS-5 phototypesetter at University of California, Berkeley.’’
- Brian W. Kernighan’s Troff Tutorial (included as USD:25) also originally contained a font sample, and while it is missing in my USENIX-printed copy, its omission appears to be an artifact of running troff manually, without a Makefile, rather than deliberate. The corresponding source file shipped on the 4.3BSD tape as */usr/doc/usd/25.trofftut/ttcharset*, and is marked ‘‘Appendix A: Phototypesetter Character Set (APS-5)’’, where the original read simply as ‘‘Appendix A: Phototypesetter Character Set’’. So it appears as though the intent was to include it in the printed manual, marked with a disclaimer that the font sample printed is that of their APS-5 *ditroff* setup, rather than the original C/A/T.
- SCCS logs for some of the documents contain entries marked with comments like ‘‘aps-5 formatting fixes’’, and the changes themselves are usually spacing issues, which often do arise as a result of slightly different font metrics.

But if the intended target was the APS-5, why do the few existing Makefiles refer to the ‘‘dp’’ *ditroff* setup? My guess is that perhaps printing on the APS was expensive/cumbersome/etc, hence they used the ‘‘dp’’ for all draft prints while editing documents. But given that they apparently used quite different *ditroff* setups for the two, methinks that ‘‘dp’’ troffings would be a poor indication of how the document would look on the APS, and if the Makefile is of no help for the latter, does it mean going back to all-manual when it’s time to submit the real thing? Thus we may never know for certain whether the masters delivered to USENIX were troffed on the APS-5 or on the ‘‘dp’’ device, but my current working hypothesis is the APS-5.

We do know, however, that the docs were run off with *some* version of *ditroff* rather than with the original C/A/T *troff* followed by a post-processor. The plain C/A/T *troff* handles characters like \(*A by substituting ‘‘equivalent’’ ASCII letters, i.e., there are no separate codes for these characters in its binary output, hence there is no way for any post-processor to recover them. However, document USD:24 contains an exhibit showing these characters, and looking at my USENIX-printed copy, I see that these capital Greek letters appear slightly different from their ASCII counterparts. That is proof that *ditroff* must have been used, rather than plain *troff* in any configuration.

1.2. Possibility of recreating the exact original appearance

There are no limits to what PostScript can do, so from the standpoint of PostScript itself there is no reason why we couldn’t recreate the original books *exactly*, i.e., such that every hyphenation, line and page break would fall in exactly the same place in the reprint as in the original. However, the real difficulty with doing that would be in the fonts. Whether the original troffing was done on an APS-5, the ‘‘dp’’ device or some other non-PostScript *ditroff* setup, an exact reproduction could, in theory, be made as follows:

1. Create custom PostScript fonts (either Type 1 or Type 3) that match those of the original troff setup in both graphic appearance and metrics;
2. Reconstruct the original troff (*ditroff*) setup, including the exact device description and font tables, and re-run the unchanged document sources through this recreated setup;
3. Write a program that takes the output from step 2 (codes for whatever historic physical output device was used, coming out of the original *ditroff* post-processor, or perhaps take the latter out of the equation and use the device-specific *ditroff* output one step earlier) and renders it in PostScript using the custom fonts from step 1.

The problem is that steps 1 and 2 both require historical information about the fonts that appears to have been lost to us through the cracks of time and obscurity. Producing custom fonts that would match the original in graphic appearance *as seen by a human eye* wouldn’t be too difficult: the visual reference in the form of my historic USENIX-printed copy would be sufficient. But the real problem is in the font *metrics*, which affects both steps 1 and 2. In order for all hyphenation, line and page breaks to fall in exactly the same places as in the original, the troff instance running the reprint needs to operate with *exactly* the same font character widths and other device information as the original historic run. (Making the output look nice would also require making the custom PostScript font design itself with these metrics in mind, but that can be a secondary polishing step.)

But the *ditroff* device and font descriptions for UCB’s APS-5 appear to have been lost, or at least I have no idea where to find them. Without this critical piece the rest of the exercise seems pointless, so I have pursued a completely different approach, using the native PostScript-targeting troff subsystem of 4.3BSD-Quasijarus which uses the standard Adobe fonts, making no attempt to emulate the exact appearance and metrics of some other historic fonts.

1.3. How this reprint has actually been made

This PostScript reprint has been produced on a real VAX machine running 4.3BSD-Quasijarus, using the new Quasijarus *pstroff* subsystem which at the time of this writing is still in the process of being polished up for release. At the heart of this *pstroff* subsystem is a new version of the *troff* program itself that is derived (purely in terms of code genealogy) from the C/A/T-targeting *troff* shipped with 4.3BSD-Tahoe, still runs under 4.3BSD, but produces PostScript output, rather than C/A/T codes or *ditroff* intermediate language. Quasijarus *pstroff* is genealogically independent of *ditroff*, but I have independently reinvented many of the same concepts, i.e., a vastly different repertoire of characters, fonts and point sizes, and thus a different internal character representation.

Pstroff is designed to format most documents using only standard Adobe fonts that are available ''for free'' in all PostScript printers; the Times family plus Symbol form the subset relevant to historic troff emulation. Some historic documents use historic troff characters that aren't available in any of the standard Adobe fonts (the *eqn*(1) page in this book uses **\(bs**, for example), and I've put a considerable amount of work into adding special emulation options to *pstroff* with which these characters become available from *pstroff*'s own special fonts, drawn to look as authentic as possible. Hence most characters come from standard Adobe fonts, but a few are supplied specially by *pstroff*.

The main difference between this PostScript reprint and the original APS-5 (presumed) print is that the Times-Roman font used for regular text in this reprint is a little more compact than its analogue used in the original print, the result being that more words fit on a line and hence on a page in this reprint than in the original. As a result, the hyphenation, line and page breaks don't line up at all.

I've spent some time thinking about whether or not this difference in font metrics and hence in where the breaks fall makes my reprint somehow ''substandard'', i.e., whether or not the font metrics and the exact location of every break should be considered an essential part of the books that **must** be reproduced exactly. I've come to the conclusion that my reprint is probably OK on philosophical grounds. The UNIX culture has always emphasized content over insignificant variations in appearance, and I have a hard time seeing the exact rendition and metrics of every character as being a fundamental part of the defining specification for troff documents. Furthermore, many of the documents were originally run off on the C/A/T (definitely at Bell Labs and possibly in the early UCB era as well), and it appears that the font metrics on the APS-5 didn't exactly match those of the C/A/T either, as evidenced by the formatting fixes in SCCS. Hence my reprint does not exactly reproduce the historical USENIX print in the minute details of the appearance, but does reproduce the content very faithfully, having been re-troffed from the document sources shipped on the 4.3BSD tape.

The manual sources from the tape have been re-troffed with 4.3BSD-Quasijarus *pstroff* by setting up a ''recipe'' consisting of a few Makefiles and scripts, such that a single *make* command produces the complete book as **urm.ps** given only a 4.3BSD-Quasijarus system, a virgin read-only copy of the original 4.3BSD */usr/man* from the tape, and the (very small) ''recipe'' package itself. The latter will be tarred up and published along with the **urm.ps** product. I can only imagine that this fully scripted and automated way must be much nicer than whatever incredibly error-prone mostly-manual production process they must have used Back In The Day, but I also can't resist the urge to point out that I am **not** using any hardware in this process that wasn't available in those days! PostScript printers and typesetters *were* already available then, and the rest is all software, written by me and running under the same good old 4.3BSD.

A similar process will follow for the remaining 5 books in the set. PRM is expected to be trivial, but many of the supplementary documents are expected to require some more work. Each volume will have its own colophon detailing every reformatting quirk that occurs in it.

This reprint of the 4.3BSD manual set and Quasijarus troff have been co-developed in parallel; as the latter is currently being polished up for release, the former effectively serves as a pre-release test. Part of the challenge has been in refining the conceptual definition of the abstract interface between troff users (document sources) and troff implementations: the ''holy grail'' is to be able to feed an unmodified historic document source to the new troff subsystem (which I define to include retuned versions of eqn and the macro packages) and have it magically produce output that is both aesthetically pleasing and conceptually ''right'', using new native PostScript ways of doing things as much as possible.

The ideal of zero changes to document sources appears to be unattainable, but I've had to think a lot about how much patching of historical sources would be acceptable and at what point it would become ''cheating''. The colophon for each volume in the manual set will contain a thorough description and discussion of any and all changes that have to be made in the reprint.

1.4. How the troff fonts looked in the USENIX version

Given that I have an original USENIX-printed set of 6 books in my possession (I've bought them on Amazon Marketplace in 2010 for use in this project as a reference) and this reprint does not attempt to reproduce the exact appearance of every character in every font visible in these books, I consider it prudent to do at least a minimal service to other troff archaeologists by giving a verbal description of how they look.

Whatever Roman, italic and bold fonts were used to typeset the USENIX-printed version, they included all 5 ligatures: **ﬁ**, **ﬂ**, **ﬀ**, **ﬃ** and **ﬄ**. Only **ﬁ** and **ﬂ** exist in standard Adobe text fonts. The running text of the USENIX-printed manuals uses all 5 ligatures, such that 'troff' appears as **troﬀ**, for example. Quasijarus *pstroﬀ* is capable of rendering these ligatures, as shown here, but they aren't standard and are drawn from a font that doesn't strictly match Times, hence they aren't used automatically. This reprint uses only **ﬁ** and **ﬂ** in running text.

One particularly pesky troff character is **\(sq**. It is generally agreed to be a hollow square in the **R** and **I** fonts, but a controversy exists as to whether **\fB\(sq\fP** is supposed to be a filled square or also hollow. In the USENIX print the latter character is a hollow square, but with slightly thicker lines than the non-bold version. (Slightly thicker meaning that one has to view them side-by-side to tell reliably which is which.) I'll address the issue further in the colophon to the USD book reprint as the character appears there more prominently.

Another pesky character is the circle **\(ci**. It appears to have been intended to line up with the square, even though the square was in the base fonts and could vary, whereas there was only one circle on the Special Mathematical Font. The two squares (regular and bold) and the circle do indeed seem to line up in the USENIX print.

The **\(em** character appears to have been rendered like in the original C/A/T: a ¾ em dash glyph centered in 1 em spacing width. (This reprint uses the standard Adobe 1 em dash character instead: —)

Curly braces { } in italicized text appear upright. There is no way to tell from the printout alone whether it's because these characters are relegated to a ''special'' font like in the C/A/T, or if there is a separate instance of each character in the italic font, but these two have been deliberately left upright like the plus symbol, for example. This reprint uses the standard Adobe *Times-Italic* font, which has italicized curly braces: *{ }*.

2. Specific notes for the URM volume

2.1. Possible differences between the USENIX print and the tape

I have caught a few differences between what appears on paper in the USENIX-printed book and what appears in the troff sources from the tape. All instances which I have caught are enumerated in this colophon. With the exception of these listed differences, I *believe* that the source files I've used to produce this reprint are fully identical in content to what's been printed in the book by USENIX.

However, the only way to prove the absence of any more unnoticed differences would be to make a direct comparison, page for page, word for word, character for character, between the extant printed book and the source files or a new print made from the latter, such as the present one. Needless to say, performing such a direct comparison on the entire body of a 500+ page book is not feasible, certainly not manually. (It *might* be possible with some advanced automated scanning and OCR technologies, but I have no knowledge of such.) Therefore, I cannot currently guarantee the absence of any more unnoticed differences.

2.2. Date string inconsistency in the footer of some man pages

Man pages *apropos*(1), *ex*(1), *lpq*(1), *more*(1), *passwd*(1) and *sysline*(1) appear in this reprint with a footer date of ''June 5, 1986''; this date string comes from the sources shipped on the 4.3BSD tape. However, the printed versions of these 6 pages in my USENIX copy of the URM book have earlier footer dates, even though there are no differences in the content.

Examination of the SCCS files reveals that in all 6 cases the differences were troff formatting fixes. While some of the fixes are ‘‘invisible’’ in the sense that one can’t tell from the printout whether the source fed to troff at that time included the fix or not, a couple of them are visible, and the versions printed in the book must have had the fixes applied. Yet the footer dates in the book predate the fixes in SCCS.

My guess is that the fixes were made by hand by whoever was in charge of preparing the manuals for USENIX, and then checked into SCCS some time later. It is worth noting though, that there is one more man page in a similar situation: *csh*(1), yet the version in my USENIX-printed book (supposedly the third printing, no idea if there were any changes between printings) shows the footer date as ‘‘June 5, 1986’’, same as in SCCS.

I haven’t been able to establish an exact cut-off date for the USENIX print, and it appears to be more of an ad hoc pick. Given the apparent lack of automation in the manual production process, it seems plausible that they could have run off the whole thing some time in the early spring of 1986, then selectively slipped in updated pages, and some updates fell through the cracks.

2.3. chfn(1) and chsh(1)

Many man pages have aliases, i.e., the set of possible valid arguments to the *man* command that will display a page is a superset of the set of distinct pages. The set printed in the book is supposed to be the latter, i.e., there is no need to waste paper by printing the same man pages several times under different names. And indeed, the manual production process weeds the aliases out.

However, two aliases have managed to slip through the cracks in a peculiar way into the USENIX URM print: *chfn*(1) and *chsh*(1), which are both aliases for *passwd*(1). It is peculiar in that while the printed *passwd*(1) page is the correct 4.3BSD version, the pages printed under *chfn*(1) and *chsh*(1) are an older version of the *passwd*(1) page (SCCS revision 6.1, for the curious). The latter two pages are identical except for the capitalized page name.

The two extraneous man pages appear both in the body of the book as well as in the table of contents, i.e., the two are in ‘‘perfect’’ agreement. I was thus faced with a dilemma as to how I should handle this situation in this reprint. My options were:

1. Replicate the erroneousness of the USENIX book by pulling *passwd.1* revision 6.1 out of SCCS and printing it as *chfn*(1) and *chsh*(1). Downside: misleading, as the *passwd* command and its mode-specific aliases *chfn* and *chsh* actually behave in 4.3BSD like the ‘‘new’’ *passwd*(1) man page says, not like the ‘‘old’’ page says.
2. Omit the two bogus pages, both from the body of the book and from the table of contents. Downside: deleting two lines from the ToC would rearrange the whole ToC. The ToC is otherwise unchanged from what came on the tape, and also identical to what’s in the USENIX book.
3. Omit the two bogus pages from the body of the book, but leave them in the ToC as historical artifacts. Downside: breaks the otherwise perfect match between the book and the ToC.

The solution I have finally settled on is rather silly: I’ve included *chfn*(1) and *chsh*(1) pages, but they are copies of the correct *passwd*(1) page with the capitalized name substituted accordingly, not the old misleading page seen in the USENIX print. This way the ToC has been left intact and it still matches the body of the book.

2.4. Table of Contents and the Permuted Index

The ‘‘head’’ section of the book before the man pages has been reprinted from the intact files shipped with 4.3BSD under */usr/man/man0*, i.e., those files have **not** been regenerated anew. This section appears to match the USENIX print as well, subject to the caveat stated in §2.1 above, and allowing for different page breaks in the verbose introduction section because of the font metric differences. However, the page breaks in the ToC and the Permuted Index appear to match exactly between this reprint and the USENIX version!

The only issue found involves the ToC for sections 5 and 8. Section 8 contains *XNSrouted*(8) and section 5 contains 6 more pages with uppercase names; all other man pages in the 4.3BSD set have all-lowercase names, as is the norm. The corresponding files from the tape (*toc5* and *toc8* under */usr/man/man0*) are sorted in the case-sensitive lexicographic order, such that the uppercase-named pages appear at the beginning (with the exception of *intro*(8) which is still moved to the very front), but in the USENIX-printed books the order of the pages everywhere (in the full ToC in the URM volume, in the partial ToCs in PRM and SMM, the actual body of section 5 in PRM and that of section 8 in SMM) is case-insensitive lexicographic: the uppercase-named pages appear in-line with the rest as if all had been preened to the same case.

Thinking about it rationally, having the pages sorted in the case-insensitive lexicographic order in the body of the book makes the most sense: a user flipping through the book looking for a page will be guided by the page headings, and those are capitalized. And since it makes the most sense for the ToC to match the body of the book exactly, I've patched the *toc5* and *toc8* files in this reprint to match the USENIX version.

My guess is that the automated process that produced these ToC files wasn't smart enough to do case-insensitive sorting, and the latter was done as a manual touch-up step. Perhaps the properly touched-up copy got troffed and sent off to USENIX, then someone re-ran the scripts and lost the touch-up.

2.5. Man pages for user-contributed software

The USENIX print of the URM book, as well as the ToC for section 1 found on the tape, includes 8 man pages which belong to user-contributed software rather than the base system, and don't appear on the 4.3BSD tape under */usr/man*: *bib*(1), *jove*(1), *mh*(1), *notes*(1), *patch*(1), *rcsintro*(1), *readnews*(1) and *vnews*(1).

I've included all 8 in this reprint. I've located their sources in the user-contributed software part of the 4.3BSD tape (*/usr/new*), and copied them over into the ''recipe'' package that drives the automated production of this PostScript reprint. I've figured out the correct incantations needed to troff them, and included them in the packaged manual section 1.

Then I've compared my reprint of these 8 pages against their appearance in the USENIX print, and found that 5 of them have had their ''SEE ALSO'' section slightly modified. I have reproduced these modifications in this reprint; see the ''recipe'' package source for the details.

2.6. eqnchar(7)

The *eqnchar*(7) man page demonstrates the use of the */usr/pub/eqnchar* file. This file can be viewed as a kind of API, part of the abstracted interface between troff users and troff implementations. In other words, a user of troff+eqn should be able to use any of the named characters listed in the *eqnchar*(7) man page and expect to get the graphic result shown there, without caring how it's actually implemented. The actual contents of the */usr/pub/eqnchar* file may differ from one troff implementation to another: the original V7 version is best fit for the C/A/T typesetter and its character set, but is a poor fit for PostScript. The Quasijarus troff implementation used to produce this reprint can implement the *eqnchar* ''API'' with much more pleasing results by substituting its own version of */usr/pub/eqnchar*.

The original *eqnchar.7* man page source file contains a verbatim copy of the original V7 */usr/pub/eqnchar* file imbedded therein. It has been modified for this PostScript reprint by replacing that imbedded copy with a **.so** line which pulls from the standard ''API'' location. Thus the graphic appearance of the man page in this printing shows how the *eqnchar* ''API'' characters are rendered by the Quasijarus troff implementation used to reprint this book.

The original V7 *eqnchar* file implemented *square* as **\(sq**, *circle* as **\(ci**, and *blot* as **\fB\(sq\fP**. The *nroff* version prints *blot* by overstriking ASCII characters **H**, **I** and **X**, so I am quite convinced that the intent was for *blot* to be a filled square, and **\fB\(sq\fP** was supposed to be a filled square as well.

As you can see from the rendition of the *eqnchar*(7) page in this printing, the Quasijarus troff implementation makes *square* a hollow square, *circle* a hollow circle, and *blot* a filled square, with matching metrics for all three. However, the implementation doesn't use **\(sq** or **\(ci**, instead the *eqnchar* characters are implemented using *pstroff* extensions which provide more native access to PostScript path construction and stroking/filling operators.

In the USENIX-printed URM book the *blot* character appears as a hollow square like *square*, but with slightly thicker lines, as that's how the historic troff implementation in question rendered the bold and non-bold versions of the **\(sq** character. But in my reprint I've chosen to render the characters as I believe they've been meant to look by their original inventors.

2.7. .bd in the troff(1) page

The *troff*(1) man page describes both *troff* and *nroff*, and the listing of command line options contains a subheading that reads ‘‘*Troff only*’’. The man page source has these words surrounded with ‘‘**.bd I 3**’’, and the intent has been to produce artificially emboldened italics.

The **.bd** request is very problematic because its numeric argument doesn't support scaling, i.e., is always taken in troff internal units or ‘‘goobies’’. The physical size of these ‘‘goobies’’ differs wildly between different troff implementations, sometimes by an order of magnitude or more. While it would be possible for a thoughtful document author to do some explicit math to compute the number in ‘‘goobies’’ from inches or ems and pass that number to **.bd**, most existing historical documents don't do that, and pass hard-coded numbers in ‘‘goobies’’ to **.bd** like this *troff*(1) man page does.

The problem is so bad that in the current version of Quasijarus *pstroff* I've chosen not to implement **.bd** at all. Because the request is simply ignored, the two words in question appear in plain italics in this reprint. I've considered setting these two words in some bold-italic or semibold-italic font, either the ‘‘free’’ Times-BoldItalic or one of the ‘‘premium’’ fonts from the full Adobe Type Library, either by patching the *troff.1* man page source or by teaching *pstroff* to do idiom recognition on **.bd** and substitute fonts behind the scenes, but decided that it would be way overkill to do any of that for just two words in a 500+ page book. Hence plain italics stay for now.

Whatever troff was used to typeset the original USENIX version did implement **.bd** ‘‘classically’’, and the two words do appear slightly emboldened in the USENIX-printed book. However, the emboldening is very slight, no more than semibold. That may be a result of that troff having smaller ‘‘goobies’’ than the C/A/T.

3. Thumb index

The USENIX printing of the 4.3BSD manual set involved some post-troff processing. All troffing was done in the classic 8.5x11" world, but then the master print must have been scaled down in post-processing, and the actual USENIX books are printed in 6x9" format. Furthermore, the copy I have features a thumb index, identifying the man page sections in URM and PRM or supplementary document numbers in the other 4 books. The thumb index is made of black print extending all the way to the right margin (and sometimes the top or bottom margin as well) of both recto and verso of each sheet making up each marked section (I've been told that such features are produced by printing on slightly larger paper, then cropping it down), with identifying text (three letters naming the volume plus the section or document number) appearing in a sans-serif font in white on black, i.e., character-shaped white holes in a flooded patch of black ink in physical terms. I reason that these decorations must have been added in some post-troff processing as no version of troff in the 1980s could have produced such white-on-black print.

I have to agree, however, that the thumb index is a very useful feature from the user's perspective, and I've made plans for reproducing it as well. The production of the present PostScript reprint is viewed as consisting of two stages, corresponding almost directly to the ‘‘troff’’ and ‘‘post-troff’’ stages that must have been done back in the 1980s. In the first stage I produce a set of PostScript files, one per volume, which are formatted with standard 8.5x11" pages, and release them free to the world via my FTP site. In a later stage I plan to tackle the problem of printing actual physical books no worse than the USENIX ones from these PostScript files, and that will involve post-processing at the PostScript level in order to be able to offer the same 6x9" format and thumb index as the USENIX predecessor.

In order to facilitate such future post-processing that will add a thumb index, the one file per volume PostScript products being produced in the present stage contain special markings. 4.3BSD-Quasijarus *pstroff* output fully conforms to the PostScript Document Structuring Conventions (DSC), and every page begins with a ‘‘%%Page:’’ DSC comment. In addition to a linear page count running the full length of every DSC-conforming PS file, the ‘‘%%Page:’’ comment allows for an arbitrary label. The **urm.ps** file for this book has these labels assigned such that they will directly assist in the future addition of the thumb index. The actual format of these labels is easier to see than to explain, just do a *grep ´^%%Page:´ urm.ps* and see for yourself. The same will hold for all other volumes in the set.

Michael Sokolov, IFCTF, SE 50 Vnt 4