Unix Programmer's Manual

Stanford University Systems Programmer's Version

This manual consists of selections from the Unix Programmer's Manual that are likely to be of use to systems programmers at Stanford. Details pertinent to system administration are not included here; they are available in the System Administrator's version. As a guide to the organization of this manual, the organization of the complete manual is outlined below. The table of contents of each volume or section is included in full, so that the reader can determine what additional material is available; portions actually included in this version are checked (except in Volume 1).

The Unix Programmer's Manual provided by Berkeley has been augmented to contain documentation of additional software used at Stanford. The complete manual consists of two volumes. Volume 1 contains brief "manual pages" describing the commands and features provided by the system. There are nine sections:

1. Commands
2. System calls
3. Subroutines
4. Special files
5. File formats and conventions
6. Games
7. Miscellaneous
8. Maintenance commands and procedures
9. PUP library routines

This systems programmer's manual contains all of Volume 1, except Section 8.

Volume 2 contains documents that supplement the manual pages in Volume 1. These are mostly articles, tutorials or manuals on specific programs, commands or systems. There are five sections:

2a and 2b	Provided by Bell Laboratories.
2c	Provided by Berkeley.
User Contributed Software	Provided by users whose software is distributed together with Unix.
Additional Material	Not part of the Berkeley manual.

This systems programmer's manual contains a variety of articles, including complete Emacs and MH manuals.

Getting Started

The following material in this manual is particularly useful for obtaining an overview of 4.2 Unix and for finding one's way around the manual:

- For users unfamiliar with Unix, the introduction to Volume 1.
- For users familiar with 4.1 BSD Unix, the documents "Changes from 4.1 BSD to 4.2 BSD Vax Unix at Stanford University", at the start of Additional Material, and "Bug fixes and changes in 4.2 BSD", at the start of Volume 2c.
- The tables of contents at the start of Volumes 1, 2, 2c, User Contributed Software and Additional Material.
- The permuted index at the start of Volume 1.

Since changes are made to the system periodically, the most reliable way to locate up-to-date documentation based on keyword is to use the command *apropos*(1) online.

UNIX PROGRAMMER'S MANUAL

4.2 Berkeley Software Distribution
Virtual VAX−11 Version

August, 1983

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

PREFACE

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
O. 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
O. 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 trademark of Bell Laboratories.
*VAX and PDP are Trademarks of Digital Equipment Corporation.

INTRODUCTION TO VOLUME 1

This volume gives descriptions of the publicly available features of the UNIX/32V† system, as extended to provide a virtual memory environment and other enhancements at U. C. Berkeley. It does not attempt to provide perspective or tutorial information upon the UNIX operating system, its facilities, or its implementation. Various documents on those topics are contained in Volume 2. In particular, for an overview see 'The UNIX Time-Sharing System' by Ritchie and Thompson; for a tutorial see 'UNIX for Beginners' by Kernighan, and for an guide to the new features of this virtual version, see 'Getting started with Berkeley Software for UNIX on the VAX' in volume 2C.

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.

The volume is divided into eight sections:

1. Commands
2. System calls
3. Subroutines
4. Special files
5. File formats and conventions
6. Games
7. Macro packages and language conventions
8. Maintenance commands and procedures

Commands are programs intended to be invoked directly by the user, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in directory */bin* (for *bin*ary programs). Some programs also reside in */usr/bin,* or in */usr/ucb,* to save space in */bin.* These directories are searched automatically by the command interpreters.

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 Section 2.

An assortment of subroutines is available; they are described in section 3. The primary libraries in which they are kept are described in *intro*(3). The functions are described in terms of C, but most will work with Fortran as well.

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

The file formats and conventions section 5 documents the structure of particular kinds of files; for 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.

Games have been relegated to section 6 to keep them from contaminating the more staid information of section 1.

---

† UNIX is a trademark of Bell Laboratories.

Section 7 is a miscellaneous collection of information necessary to writing in various specialized languages: character codes, macro packages for typesetting, etc.

The maintenance section 8 discusses commands and procedures not intended for use by the ordinary user. The commands and files described here are almost all kept in the directory */etc*.

Each section consists of a number of independent entries of a page or so each. The name of the entry is in the upper corners of its pages, 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. 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 of whose subsections will always appear.

The *name* subsection lists the exact names of the commands and subroutines covered under the entry and gives a very short description of their purpose.

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 indicate 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 '−' is often taken to mean some sort of 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 which are built into the program.

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

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

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

At the beginning of the volume 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 which 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 Volume 2 for a more complete introduction to the system.

Logging in. You must call UNIX from an appropriate terminal. Almost any ASCII terminal capable of full duplex operation and generating the entire character set can be used. You must also have a valid user name, which may be obtained, together with necessary telephone numbers, from the system administration. After a data connection is established, the login procedure depends on what kind of terminal you are using and local system conventions. The following examples are typical.

300-baud terminals: Such terminals include the GE Terminet 300, and most display terminals run with popular modems. These terminals generally have a speed switch which should be set at '300' (or '30' for 30 characters per second) and a half/full duplex switch which should 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 turns off the printer on 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.

1200- and 150-baud terminals: If there is a half/full duplex switch, set it at full-duplex. When you have established a data connection, the system types out a few garbage characters (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 in clear, then respond with your user name. From the TTY 37 terminal, and any other which has the 'newline' function (combined carriage return and linefeed), terminate each line you type with the 'new line' key, otherwise use the 'return' key.

Hard-wired terminals. Hard-wired terminals usually begin at the right speed, up to 9600 baud; otherwise the preceding instructions apply.

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 upper-case letters to lower 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), which discusses the login sequence in more detail, and *tty*(4), which discusses terminal I/O.

Logging out. There are three ways to log out:

> By typing 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 character '@' in typed input kills all the preceding characters in the line, so typing mistakes can be repaired on a single line. Also, the character '#' erases the last character typed. (Most users prefer to use a backspace rather than '#', and many prefer control-U instead of '@'; *tset*(1) or *stty*(1) can be used to arrange this.) Successive uses of '#' erase characters back to, but not beyond, the beginning of the line. '@' and '#' can be transmitted to a program by preceding them with '\'. (So, to erase '\', you need two '#'s).

The 'break' or 'interrupt' key causes an *interrupt signal,* as does the ASCII 'delete' (or 'rubout') character, 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 don't 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. Many users change this interrupt character to be ^C (control-C) using *stty*(1).

It is also possible to suspend output temporarily using ^S (control-s) and later resume output with ^Q. In a newer terminal driver, it is possible to cause output to be thrown away without interrupting the program by typing ^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.

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 will set or reset this mode. *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 indicate that it is ready for another command.

The shells have many other capabilities, which 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 Bell-labs provided shell. 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, which greatly enhance its power when used interactively. *Csh* also supports the job-control facilities; see *csh*(1) or the Csh introduction in volume 2C 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 in a hierarchy of directories. When the system administrator gave you a user name, he 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/lem/filex* refers to the file *filex* in the directory *lem; lem* 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 which 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* (in *rm*(1)) for destroying them.

For a fuller discussion of the file system, see 'The UNIX Time-Sharing System,' by Ken Thompson and Dennis Ritchie. It may also be useful to glance through section 2 of this manual, which discusses system calls, even if you don't 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), the Pascal compiler *pc*(1), and interpreter *pi*(1) and *px*(1), and the Lisp system *lisp*(1). User contributed software in the latest release of the system supports APL, the Functional Programming language, and Icon. Refer to *apl*(1), *fp*(1), and *icon*(1), respectively for more information about each. After the program text has been entered through the editor and written on 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* to move it to a less exposed name soon.)

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 *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, pr, more* and *nroff,* all in section 1.

The *cat* command simply dumps ASCII text on the terminal, with no processing at all. The *pr* command paginates the text, supplies headings, and has a facility for multi-column output. *Nroff* 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 prepares documents for a Graphics Systems phototypesetter or a Versatec Plotter; it is very similar to *nroff,* 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.

More(1) is useful for preventing the output of a command from zipping off the top of your screen. It is also well suited to perusing files.

Status inquiries. Various commands exist to provide you with useful information. *w*(1) prints a list of users presently 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) is used; *mail*(1) will leave a message whose presence will be announced to another user when he next logs in. The write-ups in the manual also suggest how to respond to the two 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.

When you log in, a message-of-the-day may greet you before the first prompt.

CONVERTING FROM THE 6TH EDITION

There follows a catalogue of significant, mostly incompatible, changes that will affect old users converting from the sixth edition on a PDP-11. No attempt is made to list all new facilities, or even all minor, but easily spotted changes, just the bare essentials without which it will be almost impossible to do anything.

Addressing files. Byte addresses in files are now long (32-bit) integers. Accordingly *seek* has been replaced by *lseek*(2). Every program that contains a *seek* must be modified. *Stat* and *fstat*(2) have been affected similarly, since file lengths are now 32- rather than 24-bit quantities.

Assembly language. This language is dead. Necromancy will be severely punnished.

Stty and *gtty.* These system calls have been extensively altered, see *ioctl*(2) and *tty*(4).

C language, lint. The syntax for initialization requires an equal sign = before an initializer, and brackets { } around compound initial values; arrays and structures are now initialized honestly. Assignment operators such as =+ and =− are now written in the reverse order: +=, −=. This removes the possibility of ambiguity in constructs such as x=−2, y=*p, and a=/*b. You will also certainly want to learn about

long integers
type definitions
casts (for type conversion)
unions (for more honest storage sharing)
#include <filename> (which searches in standard places)

The program *lint*(1) checks for obsolete syntax and does strong type checking of C programs, singly or in groups that are expected to be loaded together. It is indispensable for conversion work.

Fortran. The old *fc* is replaced by *f77,* a true compiler for Fortran 77, compatible with C. There are substantial changes in the language; see 'A Portable Fortran 77 Compiler' in Volume 2.

Stream editor. The program *sed*(1) is adapted to massive, repetitive editing jobs of the sort encountered in converting to the new system. It is well worth learning.

Standard I/O. The old *fopen, getc, putc* complex and the old *−lp* package are both dead, and even *getchar* has changed. All have been replaced by the clean, highly efficient, *stdio* package, *intro*(3S). The first things to know are that *getchar*(3) returns the integer EOF (−1) (which is not a possible byte value) on end of file, that 518-byte buffers are out, and that there is a defined FILE data type.

Make. The program *make*(1) handles the recompilation and loading of software in an orderly way from a 'makefile' recipe given for each piece of software. It remakes only as much as the modification dates of the input files show is necessary. The makefiles will guide you in building your new system.

Shell, chdir. F. L. Bauer once said Algol 68 is the Everest that must be climbed by every computer scientist because it is there. So it is with the shell for UNIX users. Everything beyond simple command invocation from a terminal is different. Even *chdir* is now spelled *cd*. If you wish to use *sh* (as opposed to *csh*) then you will want to study *sh*(1) long and hard.

C shell. *Csh*(1), developed at Berkeley, has features comparible to *sh*. It includes a history mechanism that saves you from retyping all or part of previous commands, as well as an efficient aliasing (macro) mechanism. The job control facilities of the system, which make the system much more pleasant to use, are currently available only with *csh*. See *csh*(1) for a description. These features make *csh* pleasant to use interactively. *Csh* programs have a syntax reminiscent of *C*, while *sh* command programs have a syntax reminiscent of ALGOL-68.

Debugging. *Sdb* is a far more capable replacement for the debugger *cdb*, and debugs C and Fortran at the source level. For machine language debugging, *adb* replaces *db*. The first-time user should be especially careful about distinguishing / and ? in *adb* commands, and watching to make sure that the *x* whose value he asked for is the real *x*, and not just some absolute location equal to the stack offset of some automatic *x*. You can always use the 'true' name, *_x*, to pin down a C external variable.

Dsw. This little-known, but indispensable facility has been taken over by *rm −ri*.

Boot procedures. Needless to say, these are all different. See section 8 of this volume, and the other documentation you should have received with your tape.

CONVERTING FROM THE DECEMBER, 1979 BERKELEY DISTRIBUTION

There have been a number of significant changes and improvements in the system. This list just gives the bare essentials:

C language changes. The C compiler now accepts and checks essentially arbitrary length identifiers and preprocessor names. There is a new type available in type casts: **void** which signifies that a value is to be ignored. It is useful in keeping lint happy about values which are not used (especially values returned from procedures). Finally, the language has been changed so that field names need not be unique to structures; on the other hand, the compiler insists that you be more honest about types involved in pointer constructs or it will warn you.

Object file format. The object file format has been changed to include a string table, so that language compilers may have names longer than 8 characters in their resulting *a.out* files. Old *.o* files must be recreated. *A.out* files will still run on both this and the December 1979 version of the system; only the symbol tables are incompatible.

Archive format and table of contents. The archive format has been changed to one which is portable between the VAX and other machines (e.g. the PDP-11). Old VAX archives should be converted with *arcv*(8); loader archives should just be recreated since the object files are also obsolete. Loader archives should have table-of-contents added by *ranlib*(1); if they dont the loader will gripe when they are used.

New tty driver, job control facilities and csh. Hand in hand are new job control facilities, a new tty driver and a new version of the C shell which supports and uses all of this. See *tty*(4) and *csh*(1) for a quick introduction.

Pascal compiler. There is a true Pascal compiler, *pc*(1) which allows separate compilation as well as mixing in of FORTRAN and C code.

Error analyzer. There is an error analyzer program *error*(1), which takes a set of error message and merges them back into the source files at the point of error. It can be used interactively to avoid inserting errors which are uninteresting. This program eliminates once and for all making lists of errors on small scraps of paper.

Mail forwarding. The system now provides mail forwarding and distribution facilities. Group and aliases are defined in the file */usr/lib/aliases* see *aliases*(5). If you change this file you will have to rerun *newaliases*(1). For any particular system a table in the source of the *delivermail* postman program may have to be changed so that it knows about the gateways on the local

machine.

System bootstrap procedures. These are totally changed; the system performs automatic reboots and preens the disks automatically at reboot. You should reread the appropriate pages in section 8 if you deal with system reboots.

CONVERTING FROM THE JUNE, 1981 BERKELEY DISTRIBUTION

Many many changes have been made. This list indicates those which are most visible to users.

Directory format. Directory entries are no longer fixed length. This forces user programs which read directories to be modified to use the *directory*(3) package.

Signals. A new signal package has replaced the previous signal mechanism as well as the "jobs library". When using the compatible *signal*(3C) interface routine, the two most important changes are: signal handlers are not reset to SIG_DFL when a process receives a signal, and while a signal handler is processing a signal, that signal is blocked until the handler returns. This has implications, in particular, for programs which process the suspend character typed at the terminal. Refer to *sigvec*, *sigblock*, *sigpause*, *sigstack*, and *sigsetmask*(2) for information about the new signal facilities.

File and path names. File names may now be up to 255 characters in length. Path names are restricted to be at most 1024 characters. These two constants are provides as MAXNAMLEN and MAXPATHLEN in <*sys/dir.h*> and <*sys/param.h*>, respectively.

System time. System time is provided in microsecond precision with 10 millisecond accuracy. The new system call *gettimeofday*(2) supplants the old *time*(3) call which is now a library routine. The major impact of this change is that programs are now written in a fashion which is independent of the line clock frequency.

Groups. A user may now be in many groups simultaneously. This has obviated the need for the *newgrp* command. See *getgroups*(2) for more information.

Stat and fstat return value. The structure returned by the *stat* and *fstat* system calls is now larger. This is due to inode numbers growing to 32-bits, time stamps expanding to 64-bits and other information being included in the return value. Consult *stat*(2) for more information.

Mail forwarding. The system now provides general internetwork mail forwarding and distribution facilities. The *sendmail*(8) program replaces the old *delivermail* facility.

Debuggers. The previous C source language debugger, *sdb*, has been replaced by a new one, *dbx*(1). *Adb*(1) has been extended to simplify debugging of the operating system.

Networking support. Many new user programs provide access to the networking facilities. The *rlogin*(1C) and *rsh*(1C) programs are intended for communicating between UNIX systems. The *telnet*(1C) and *ftp*(1C) programs support the DARPA Internet standard protocols. The *netstat*(1) program is useful in watching network activity.

TABLE OF CONTENTS

1. Commands and Application Programs

2. System Calls

3. C Library Subroutines

3F. Fortran Library

3M. Math Library

3N. Internet Network Library

3S. C Standard I/O Library Subroutines

3X. Other Libraries

3C. Compatibility Library Subroutines

4. Special Files

5. File Formats

6. Games

7. Miscellaneous

8. System Maintenance

9. Pup library

PERMUTED INDEX