UNIX PROGRAMMER'S MANUAL

Third Edition

K. Thompson

D. M. Ritchie

February, 1973

PREFACE TO THE THIRD EDITION

In the months since the last appearance of this manual, many changes have occurred both in the system itself and in the way it is used.

Perhaps most obviously, there have been additions, deletions, and modifications to the system and its software. It is these changes, of course, that caused the appearance of this revised manual.

Second, the number of people spending an appreciable amount of time writing UNIX software has increased. Credit is due to L. L. Cherry, M. D. McIlroy, L. E. McMahon, R. Morris, J. F. Ossanna, and E. N. Pinson for their contributions.

Finally, the number of UNIX installations has grown to 16, with more expected. None of these has exactly the same complement of hardware or software. Therefore, at any particular installation, it is quite possible that this manual will give inappropriate information.

In particular, *any system which uses a PDP-11/20 processor will not include all the software described herein, nor will the software behave the same way.* The second, or even the first, edition of this manual is likely to be more appropriate.

Besides additions, deletions, and modifications to the writeups in each section, this manual differs from its predecessors in two ways: all the commands used for system maintenance and not intended for normal users have been moved to a new section VIII; and there is a new "How to Get Started" chapter that gives some elementary facts and many pointers to other sections.

INTRODUCTION TO THIS MANUAL

This manual gives descriptions of the publicly available features of UNIX. It provides neither a general overview (see "The UNIX Time-sharing System" for that) nor details of the implementation of the system (which remain to be disclosed).

Within the area it surveys, this manual attempts to be as complete and timely as possible. A conscious decision was made to describe each program in exactly the state it was in at the time its manual section was prepared. In particular, the desire to describe something as it should be, not as it is, was resisted. Inevitably, this means that many sections will soon be out of date. (The rate of change of the system is so great that a dismayingly large number of early sections had to be modified while the rest were being written. The unbounded effort required to stay up-to-date is best indicated by the fact that several of the programs described were written specifically to aid in preparation of this manual!)

This manual is divided into eight sections:

    I.   Commands
    II. System calls
    III. Subroutines
    IV. Special files
    V.   File formats
    VI. User-maintained programs
    VII. Miscellaneous
    VIII. Maintenance

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 binary programs). This directory is searched automatically by the command line interpreter. Some programs classified as commands are located elsewhere; this fact is indicated in the appropriate sections.

System calls are entries into the UNIX supervisor. In assembly language, they are coded with the use of the opcode "sys", a synonym for the trap instruction.

A small assortment of subroutines is available; they are described in section III. The binary form of most of them is kept in the system library /usr/lib/liba.a.

The special files section IV discusses the characteristics of each system "file" which actually refers to an I/O device. Unlike previous editions, the names in this section refer to the DEC device names for the hardware, instead of the neames of the special files themselves.

The file formats section V 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.

User-maintained programs (section VI) are not considered part of the UNIX system, and the principal reason for listing them is to indicate their existence without necessarily giving a complete description. The author should be consulted for information.

The miscellaneous section (VII) gathers odds and ends.

Section VIII discusses commands which are not intended for use by the ordinary user, in some cases because they disclose information in which he is presumably not interested, and in others because they per-

form privileged functions.

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, its preparation date in the upper middle. Entries within each section are alphabetized. The page numbers of each entry start at 1. (The earlier hope for frequent, partial updates of the manual is clearly in vain, but in any event it is not feasible to maintain consecutive page numbering in a document like this.)

All entries have a common format.

The name section repeats the entry name and gives a very short description of its purpose.

The synopsis summarizes the use of the program being described. A few conventions are used, particularly in the Commands section:

Underlined 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 flag 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 section discusses in detail the subject at hand.

The files section gives the names of files which are built into the program.

A see also section gives pointers to related information.

A diagnostics section discusses the diagnostics that may be produced. This section tends to be as terse as the diagnostics themselves.

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

Previous edition of this manual had an owner section, which has been dropped from this edition because the "owners" of many routines became fairly hard to pin down. The major contributors to UNIX, (cast in order of appearance) together with their login names and most notable contributions, are

ken	K. Thompson	(UNIX, many commands)
dmr	D. M. Ritchie	(many commands, as, ld, C)
jfo	J. F. Ossanna	(roff, nroff)
doug	M. D. McIlroy	(tmg, m6)
rhm	R. Morris	(dc, much of library)
lem	L. E. McMahon	(cref)
llc	L. L. Cherry	(form, fed, salloc)
csr	C. S. Roberts	(tss)
enp	E. N. Pinson	(proof)

At the beginning of this document 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.

This manual was prepared using the UNIX text editor ed and the formatting program roff.

The assistance of R. Morris is gratefully acknowledged.

HOW TO GET STARTED

This section provides 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.

Logging in

You must call UNIX from an appropriate terminal. UNIX supports ASCII terminals typified by the TTY 37, the GE Terminet 300, the Memorex 1240, and various graphical terminals on the one hand, and IBM 2741-type terminals on the other.

To use UNIX, you must have a valid UNIX user name, which may be obtained, together with the telephone number, from the system administrators.

The same telephone number serves terminals operating at all the standard speeds. After a data connection is established, the login procedure depends on what kind of terminal you are using.

TTY 37 terminal

UNIX will type out "login: "; you respond with your user name. From the TTY 37 terminal, and any other which has the "new-line" function (combined carriage return and linefeed), terminate each line you type with the "new line" key (not the "return" key).

300-baud terminals

Such terminals include the GE Terminet 300, most display terminals, Execuport, TI, and certain Anderson-Jacobson terminals. 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. (Note that this switch will often have to be changed since MH-TSS requires half-duplex). When a connection with UNIX is established, a few garbage characters are typed (the login message at the wrong speed). Depress the "break" key; this is a speed-independent signal to UNIX that a 300-baud terminal is in use. UNIX will type "login: " at the correct speed; you type your user name, followed by the "return" key. Henceforth, the "return", "new line", or "linefeed" keys will give exactly the same results. Each line must be terminated with one of these keys; no one is listening to you until the return is received.

Selectric terminals

From an IBM 2741 or the Anderson-Jacobson Selectric terminal, no message will appear. After the data connection is established, press the "return" key. UNIX should type "login: " as described above. If the greeting does not appear after a few seconds, unlock the keyboard by switching the terminal to local and back to remote, and type "return". If necessary, hang up and try again; something has gone wrong.

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 UNIX program, the Shell, will type a "%" to you. (The Shell is described below under "How to run a program".

For more information, consult getty(VII), which discusses the login sequence in more detail, and dc(IV), which discusses typewriter I/O.

Logging out

There are three ways to log out:

> You can simply hang up the phone. Hanging up is safe if you are at command level, that is, if the Shell has just typed its prompt signal "%". It is also safe if you are in interactive system programs, for example the editor. It is unsafe if you are executing a non-interactive program, or one of your own programs, which either does not read the typewriter or ignores the end-of-file indications which will result from hanging up. The reason is that UNIX, unlike most systems, does not terminate a program simply because it has been hung-up upon.

> You can 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 also log in directly as another user by giving a login command (login (I)).

How to communicate through your terminal

When you type to UNIX, a gnome deep in the system is gathering your characters and saving them in a secret place. The characters will not be given to a program until you type a return, as described above in *Logging in.*

UNIX typewriter I/O is full-duplex (except for Selectric terminals). 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 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 stops echoing input characters, and starts echoing "#" no matter what you typed. The last character which was echoed correctly will be received correctly by the program to which you were talking; subsequent characters have been thrown away.

On a typewriter input line, the character "@" kills all the characters typed before it, so typing mistakes can be repaired on a single line. Also, the character "#" erases the last character typed. 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 ASCII "delete" (a.k.a. "rubout") character is not passed to programs but instead generates an *interrupt signal.* 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.

The quit signal is generated by typing the ASCII FS character. 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 "new line" function or whether it must be simulated with carriage-return and line-feed. In the latter case, all input carriage returns are turned to new-line 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 stty command (I) 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 stty command (I) will set or reset this mode. Also, there is a file which, if printed on TTY 37 or TermiNet 300 terminals, will set the tab stops correctly (tabs(VII)).

Section dc(IV) discusses typewriter I/O more fully. Section kl(IV) discusses the console typewriter.

How to run a program; The Shell

When you have successfully logged into UNIX, a program called the 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 first in your current directory (see next section) for a program with the given name, and if none is there, then in a system directory. 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 "%" at you to indicate that it is ready for another command.

The Shell has many other capabilities, which are described in detail in section sh(I).

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 permissions 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 destruction, let alone perusal, by other users.

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

Path names

To reference 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. E.g.: "/usr/lem/filex" refers to file "filex" in directory "lem"; "lem" is itself a sub-directory 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 sub-directory (no prefixed "/").

Without important exception, a path name may be used anywhere a file name is required.

Important commands which modify the contents of files are cp(I), mv(I), and rm(I), which respectively copy, move (i.e. rename) and remove files. To find out the status of files or directories, use ls(I) and stat(I).

See mkdir(I) for making directories; rmdir(I) for destroying them.

For a fuller discussion of the file system, see MM-71-1273-4. It may also be useful to glance through section II of this manual, which discusses system calls, even if you don't intend to deal with the system at the assembly-language level.

Writing a program

To enter the text of a source program into a UNIX file, use ed(I). The three principal languages in UNIX are assembly language (see as(I)), Fortran (see fc(I)), and C (see cc(I)). 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.) If you wrote in assembly language, you will probably need to load the program with library subroutines; see ld(I). The other two language processors call the loader automatically.

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 "%" prompt.

The next command you will need is db(I). As a debugger, db is better than average for assembly-language programs, marginally useful for C programs (when completed, cdb(I) will be a boon), and virtually useless for Fortran.

Your programs can receive arguments from the command line just as system programs do. For assembly language programs, see exec(II).

Text processing

Almost all text is entered through the editor. The commands most often used to write text on a terminal are: cat(I), pr(I), roff(I), or nroff(I).

The cat command simply dumps ASCII text on the terminal, with no processing at all. The pr command paginates the text and supplies headings. The nroff command is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file. The roff command is a somewhat less elaborate formatting program, and requires somewhat less forethought.

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(I) is used. To leave a message the presence of which will be announced to another user when he next logs in, mail(I) is used. The write-ups in the manual also suggest how to respond to the two commands if you are a target.

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