PWB/UNIX

User's Manual

Edition 1.0

T. A. Dolotta
R. C. Haight
E. M. Piskorik

Editors

May 1977

Bell Telephone Laboratories, Incorporated

This manual was set on a Graphic Systems, Inc. phototypesetter driven by the TROFF formatting program operating under the PWB/UNIX system. The text of the manual was prepared using the ED text editor.

ACKNOWLEDGEMENTS

The form and organization of this manual, as well as a major fraction of its contents, have been copied from the *UNIX Programmer's Manual—Sixth Edition*, by K. Thompson and D. M. Ritchie (Bell Telephone Laboratories, May 1975). The number of our colleagues who have contributed to UNIX and PWB/UNIX software and documentation is, by now, too large to list here, but the usefulness and acceptance of UNIX and of PWB/UNIX is a true measure of their collective success.

Piscataway, New Jersey T.A.D.
May 1977 R.C.H.
E.M.P.

INTRODUCTION

This manual describes the features of PWB/UNIX. It provides neither a general overview of UNIX (for that, see ''The UNIX Time-Sharing System,'' *Comm. ACM* **17**(7):365-75, July 1974, by D. M. Ritchie and K. Thompson), nor details of the implementation of the system.

This manual is divided into eight sections:

I. Commands and Application Programs
II. System Calls
III. Subroutines
IV. Special Files
V. File Formats and Conventions
VI. Games
VII. Miscellaneous
VIII. System Maintenance

Section I (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory */bin* (for *bin*ary programs). Some programs also reside in */usr/bin*, to save space in */bin*. These directories are searched automatically by the command interpreter called the Shell.

Section II (*System Calls*) describes the entries into the UNIX supervisor, including the assembler and the **C** language interfaces. In the assembler, these system calls are invoked by the *sys* operation code, which is a synonym for the *trap* instruction.

Section III (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in directory */lib*. The subroutines available for **C** and for Fortran are also included there; they reside in */lib/libc.a* and */lib/libf.a*, respectively.

Section IV (*Special Files*) discusses the characteristics of each system ''file'' that actually refers to an input/output device. The names in that section refer to the Digital Equipment Corporation's device names for the hardware, instead of the names of the special files themselves.

Section V (*File Formats and Conventions*) documents the structure of particular kinds of files; for example, the form of the output of the assembler and the loader is given. Excluded are files used by only one command, for example, the assembler's intermediate files.

Section VIII (*System Maintenance*) discusses commands that are not intended for use by the ordinary user, in some cases because they disclose information in which he or she is presumably not interested, and in others because they perform 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. Entries within each section are alphabetized. The page numbers of each entry start at 1.

All entries are based on a common format, not all of whose parts always appear:

The NAME part repeats the name of the entry and states (very briefly) its purpose.

The SYNOPSIS part summarizes the use of the program being described. A few conventions are used, particularly in Section I (*Commands*):

Boldface strings are considered literals, and are to be typed just as they appear (they are usually underlined in the typed version of the manual entries *unless* they are juxtaposed with an *italic* string).

Italic strings usually represent substitutable arguments (they are underlined in the typed version of the manual entries).

Square brackets ‘‘[ ]’’ around an argument indicate that the argument is optional. When an argument is given as ‘‘name’’ or ‘‘file’’, it always refers to a *file* name.

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 ‘‘–’’ or a plus sign ‘‘+’’ is often taken to be 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 ‘‘–’’ or ‘‘+’’.

The DESCRIPTION part discusses in detail the subject at hand.

The FILES part gives the file names that are built into the program.

The SEE ALSO part gives pointers to related information.

The DIAGNOSTICS part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The BUGS part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents (organized by section and alphabetized within each section) and a permuted index derived from that table precede Section I. Within each *index* entry, the title of the *manual* entry 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.

All manual entries are available on-line via the *man*(I) command (q.v.).

HOW TO GET STARTED

This section provides the basic information you need to get started on UNIX (we will use ''UNIX'' in this section to mean both ''UNIX'' and ''PWB/UNIX'', unless the distinction matters): how to log in and log out, how to communicate through your terminal, and how to run a program. See *UNIX for Beginners* by B. W. Kernighan for a more complete introduction to the system.

Logging in. You must call UNIX from an appropriate terminal. UNIX supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained, together with the telephone number, from the system administrator. The same telephone number serves terminals operating at speeds of 110, 150, and 300 baud. After a data connection is established, the log in procedure depends on the kind of terminal you are using.

> *300-baud terminals:* These terminals generally have a speed switch that should be set to ''300'' (or ''30'', for 30 characters per second) and a half-/full-duplex switch that should be set to full-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 (and you should!), the system asks for it, but does not print (''echo'') it on the terminal. After you have logged in, the ''return'', ''new-line'', and ''line-feed'' keys will give exactly the same result.
>
> *TELETYPE*® Model 37: 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 150-baud terminal is in use. The system then will type ''login:'', this time at 150 baud (another ''break'' at this point will get you down to 110 baud); you respond with your user name. From the TELETYPE Model 37, and any other terminal that has the ''new-line'' function (combined ''carriage-return'' and ''line-feed'' pair), terminate each line you type with the ''new-line'' key (*not* the ''return'' key).

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 input to lower case. When you have logged in successfully, the Shell program will type a ''%'' to you. (The Shell is described below under *How to run a program*.)

For more information, consult *login*(I) and *getty*(VIII), which discuss the login sequence in more detail, and *tty*(IV), which discusses terminal input/output. See *terminals*(VII) for information about various terminals.

Logging out. There are three ways to log out:

> You can simply hang up the phone.
>
> You can log out by typing an end-of-file indication (ASCII EOT character, usually typed as ''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.

How to communicate through your terminal. When you type to UNIX, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a ''return'' (or ''new-line''), as described above in *Logging in.*

UNIX terminal input/output 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 output will have the input characters interspersed. However, whatever you type will be saved 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.

On a terminal input line, the character ''@'' kills all the characters typed before it, so typing mistakes can be repaired on a single line. 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 ''\''. (Thus, 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*, just like the ''break'', ''interrupt'', or ''attention'' 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 *ed*(I), 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 a ''carriage-return'' and ''line-feed'' pair. In the latter case, all *input* ''carriage-return'' characters are changed to ''line-feed'' characters (the standard line delimiter), and a ''carriage-return'' and ''line-feed'' pair is echoed to the terminal. If you get into the wrong mode, the *stty*(I) 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 tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty*(I) command will set or reset this mode. The system assumes that tabs are set every eight columns. The *tabs*(I) command will set tab stops on your terminal, if that is possible.

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. Normally, the Shell looks first in your current directory (see *The current directory* below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the Shell can find them. The command name is always the first word on an input line to the Shell; it and its arguments are separated from one another by space or tab characters.

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 *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 or she 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 assumed to be 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 respective owners. As a matter of observed fact, many UNIX users do not 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 refer to files not in the current directory, you must use a path name. Full path names begin with ‘‘/’’, which is 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 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 corresponding subdirectory (without a prefixed ‘‘/’’).

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

Important commands that 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). See *mkdir*(I) for making directories and *rmdir*(I) for destroying them.

For a fuller discussion of the file system, see ‘‘The UNIX Time-Sharing System’’ (*Comm. ACM* **17**(7):365-75, July 1974) by D. M. Ritchie and K. Thompson. 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 that level.

Writing a program. To enter the text of a source program into a UNIX file, use *ed*(I). The three principal languages available under UNIX are **C** (see *cc*(I)), Fortran (see *fc*(I)), and assembly language (see *as*(I)). After the program text has been entered through the editor and written in a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file in the current directory named ‘‘a.out’’. (If the output is precious, use *mv*(I) 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.

Next, you will need *cdb*(I) or *db*(I) to examine the remains of your program. The former is useful for **C** programs, the latter for assembly-language. No debugger is much help for Fortran.

Your programs can receive arguments from the command line just as system programs do. See *exec*(II).

Text processing. Almost all text is entered through the editor *ed*(I). The commands most often used to write text on a terminal are: *cat*(I), *pr*(I), and *nroff*(I). The *cat*(I) command simply dumps ASCII text on the terminal, with no processing at all. The *pr*(I) command paginates the text, supplies headings, and has a facility for multi-column output. *Nroff*(I) is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file; it produces output on a typewriter-like terminal. *Roff*(I) is a less elaborate text formatting program, and requires somewhat less forethought; it is obsolescent. *Troff*(I) is similar to *nroff*(I), but drives a Graphic Systems, Inc. phototypesetter. It was used to typeset this manual.

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; *mail*(I) will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

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