HOW TO GET STARTED

This discussion provides the basic information you need to get started on UNIX (we will use "UNIX" here to mean both "UNIX" and "CB-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 dial up 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 of the UNIX system, from the administrator of your system. Usually, the same telephone number serves terminals operating at speeds of 110, 150, and 300 baud. A different number may be used for 1200-baud service. After a data connection is established, the *login* 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:** and you then 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. The system may prompt you for a dialup password which is established by the system administrator (you need to know it to use any dial port into the system). After you have logged in, the "return", "new-line", and "line-feed" keys will give exactly the same result.

Higher-speed terminals: Terminals designed to run at higher data rates than 300 baud (i.e., 1200 baud) can be utilized in full-duplex mode provided *input* remains character-by-character, typing speed.

TELETYPE® Model 37 (and other terminals less than 300 baud): 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. At this point the system will prompt you for any necessary passwords (see *300-baud terminals* above). 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).

Non-dial Terminals: In this case, the terminal should be prompting with the message "LOGIN:", if it is not, typing a "return" will usually cause it to do so. If it types garbage back at you, it is probably at the wrong speed; typing "break" will cause UNIX to cycle the terminal to another speed. If you try a few "breaks" and "returns" with no luck, see your local system guru or administrator - maybe the terminal is out of order.

It is important that you type your login name in lower case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and that you mean all subsequent upper-case input to be treated as lower case. When you have logged in successfully, the shell will type a $ to you. (The shell is described below under *How to run a program.*)

For more information, consult *login*(1) and *getty*(1M), which discuss the login sequence in more detail, *tty*(4), which discusses terminal input/output, and *stty*(1), which tells you how to describe the characteristics of your terminal to the system (*profile*(5) explains how to accomplish this last task automatically every time you log in).

Logging out. There are four 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 (on some dial-up lines the line will be hung-up without the "LOGIN:" message appearing).

You can also log in directly as another user by giving a *login* command.

You can sit around for a while. After a specified interval has elapsed with no activity on your part you will be automatically logged out. The default wait interval is specified by the system administrator when the system is built.

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 interspersed in it the input characters. However, whatever you type will be saved and interpreted in the 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 an input line from a terminal, 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 # will erase characters back to, but not beyond, the beginning of the line; @ and # can be typed into a program by preceding them with \ (thus, to erase a \, you need two #s).

The ASCII DC3 (control-s) character can be used to temporarily stop output. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed only when an ASCII DC1 (control-q) is typed. These start/stop characters are not passed to any other program when used in this manner. On CB-UNIX only, output may also be stopped by typing the break or escape keys. In this case, typing another escape (or any other characters, for that matter) will cause output to be resumed.

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 *ed*(1), 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 as to 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*(1) command will rescue you. *Stty* can also be used to change the default *erase* and *kill* characters mentioned above.

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*(1) command will set or reset this mode. The system assumes that tabs are set every eight character positions.

How to run a program. When you have successfully logged into UNIX, a program called the shell is listening to your terminal. The shell reads the lines you type, splits them into a command name and its 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 usually the first word on an input line to the shell; the command and its arguments are separated from one another by space and/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*(1).

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, and known as your *login* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is by default assumed to be in this directory. Because 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, or by the system administrator. To change the current directory 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 /, 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/ae/filex** refers to file **filex** in directory **ae**, while **ae** is itself a subdirectory of **usr**, and **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*(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). Use *mkdir*(1) for making directories and *rmdir*(1) for destroying them.

For a fuller discussion of the file system, see the references cited at the beginning of the *INTRODUCTION* above. 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 *ed*(1). The four principal languages available under UNIX are C (see *cc*(1)), Fortran (see *f77*(1)), bs (a compiler/interpreter in the spirit of Basic, see *bs*(1)), and assembly language (see *as*(1)). After the program text has been entered with the editor and written into 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 that output is precious, use *mv*(1) to give it a less vulnerable name). If the program is written in assembly language, you will probably need to load with it library subroutines (see *ld*(1)). Fortran and C call the loader automatically; programs written in *bs*(1) are interpreted and, therefore, do not need to be loaded.

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.

If any execution (run-time) errors occur, you will need *adb*(1) to examine the remains of your program.

Your programs can receive arguments from the command line just as system programs do. See *exec*(2).

Text processing. Almost all text is entered through the editor *ed*(1). The commands most often used to write text on a terminal are *cat*(1), *pr*(1), and *nroff*(1). The *cat*(1) command simply dumps ASCII text on the terminal, with no processing at all. The *pr*(1) command paginates the text, supplies headings, and has a facility for multi-column output. *Nroff*(1) 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. *Troff*(1) is very similar to *nroff*(1), but drives a Graphic Systems, Inc. phototypesetter. (It was used to typeset this manual.) There are several "macro" packages (especially the so-called *mm* package) that significantly ease the effort required to use *nroff*(1) and *troff*(1); Section 5 entries for these packages indicate where you can find their detailed descriptions.

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 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 $.

Normally, the UNIX system runs in a smooth manner and users need not be concerned about details of system operation; however, when a hardware problem exists or when new features are being tested, users may be asked to log off so that problems can be corrected. If asked to log off please do so promptly as work done may be in danger of being lost or destroyed. Whenever possible, the system will give sufficient warning of an impending outage so that you may "gracefully" log off.