
Introduction to the UNIX Operating System on IT Systems

This document is intended to introduce you to the UNIX operating system. It will provide you with a basic understanding of the UNIX operating system, its file and directory structure, basic UNIX commands, and how to get on-line help. The information in this document is primarily designed for (but not limited to) Sun workstations configured and maintained by Information Technology, such as those used in the Owl-net network.

If you are using a workstation that is configured to run exclusively as an X display server, you may wish to acquire the document, UNIX 2, Introduction to X Windows.



Table of Contents

Introduction.....	4
Font Conventions.....	4
What is UNIX?	4
UNIX Layers.....	5
Basic UNIX Elements.....	5
Logging In and Out.....	6
Getting the Login Prompt	6
TTY Terminal	6
Workstation.....	6
Entering Your Userid and Password.....	7
Logging Out	7
Workstations and TTY Terminals	7
X Terminals	8
Changing Your Password	8
UNIX Commands	9
The UNIX Shell	9
About UNIX Commands	9
Redirecting Input and Output.....	10
Getting On-Line Help with Commands	10
Case Sensitivity	11
Special Keys and Control Characters.....	11
A Selected Command List	12
Setup and Status Commands.....	12
File and Directory Commands	13
Editing Tools.....	13
Formatting and Printing Commands.....	13
Program Controls, Pipes, and Filters	14
Other Tools and Applications	14
About UNIX Files	15
Creating Files	15
Displaying Files	16

Listing Files	16
Copying Files	17
Renaming Files	17
Deleting Files	18
Creating Links Between Files	18
Printing Files	19
Directories.....	19
About UNIX Directories.....	19
Displaying Directories	20
Changing Directories	21
Moving Files Between Directories	21
Copying Files to Other Directories	22
Renaming Directories	22
Removing Directories	22
File and Directory Permissions.....	22
Processes.....	27
Viewing Your Processes	27
Running Background Jobs	28
Process Scheduling Priority	28
Remote Login	29
Troubleshooting	29
Logging In.....	29
Logging Out.....	30
Additional Resources.....	30
Problems or Questions.....	31
Faculty, Staff, and Graduate Students:	31
Undergraduates:.....	31

Introduction

This document is only a brief introduction to UNIX and does not include information on how to use all of its capabilities.

Since this introduction to UNIX is brief, we encourage you to seek out more detailed introductory information. Information Technology offers free Short Courses on UNIX that you can sign up for. Fondren Library has a number of very good introductory books on UNIX available for checkout. These books go into more detail and cover more of the operating system than will be covered in this document. You can also purchase some introductory UNIX books at the Rice Campus Store or almost any local bookstore with a computing section.

This document introduces you to the basics of UNIX, including:

- How to get started
- Shell and Commands
- File structures
- Directories

There are several variants of UNIX on the Rice campus; the type used most on Information Technology supported networks is from Sun Microsystems. Therefore, although the information concerning the UNIX operating system presented here is not limited to Sun workstations, it will focus on them.

Information Technology maintains groups of machines for different groups of users; these groups of machines are known as *domains*. The major Information Technology domains are OwlNet, the Rice UNIX Facility (RUF), and Information Technology administration. Other departments or divisions may have their own domains, such as Computer Science or Electrical and Computer Engineering. Accounts in one domain are not valid on machines of other domains. Every workstation is labeled with its name and domain. If you have an OwlNet account, use the workstations or terminals in A121 or B223 Abercrombie, 241 Mechanical Engineering, 102 Ryon Lab, 105 Mudd, 221 Physics, or the workstations at your residential college.

Font Conventions

This document uses the true type font setting to represent any feedback echoed on the computer screen after you type a command. When you are told to type a command, this command will be printed in Times Roman font.

```
this is the true type font.
```

What is UNIX?

UNIX is a powerful computer operating system originally developed at AT&T Bell Laboratories. It is very popular among the scientific, engineering, and academic communities due to its multi-user and multi-tasking environment, flexibility and portability, electronic mail and networking capabilities, and the numerous programming, text processing and scientific utilities available. It has also gained widespread acceptance in government and business. Over the years, two major forms (with several vendor's variants of each) of UNIX have evolved: AT&T UNIX System V and the Univer-

sity of California at Berkeley's Berkeley Software Distribution (BSD). This document will be based on the SunOS 4.1.3_U1, Sun's combination of BSD UNIX (BSD versions 4.2 and 4.3) and System V because it is the primary version of UNIX available at Rice. Also available are Solaris, a System V-based version, and IRIX, used by Silicon Graphics machines.

UNIX Layers

When you use UNIX, several layers of interaction are occurring between the computer hardware and you. The first layer is the *kernel*, which runs on the actual machine hardware and manages all interaction with the hardware. All *applications* and *commands* in UNIX interact with the kernel, rather than the hardware directly, and they make up the second layer. On top of the applications and commands is the command-interpretor program, the *shell*, which manages the interaction between you, your applications, and the available UNIX commands. Most UNIX commands are separate programs, distinct from the kernel. A final layer, which may or may not be present on your system, is a *windowing system* such as X. The windowing system usually interacts with the shell, but it can also interact directly with applications. The final "layer" is you, the user. You will interact with the entire operating system through just the shell, or through a combination of the shell and the window system. The figure below gives a visual representation of the layers of UNIX.

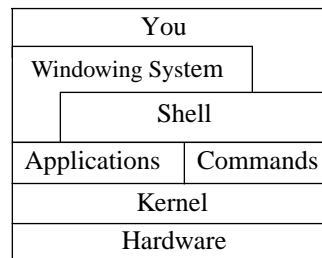


FIGURE 1. UNIX Layers

Basic UNIX Elements

You need to be familiar with six basic elements of UNIX. They are: *commands*, *files*, *directories*, *your environment*, *processes*, and *jobs*. *Commands* are the instructions you give the system to tell it what to do. *Files* are collections of data that have been given *filenames*. A file is analogous to a container in which you can store documents, raw data, or programs. A single file might contain the text of a research project, statistical data, or an equation processing formula. Files are stored in *directories*. A directory is similar to a file cabinet drawer that contains many files. A directory can also contain other directories. Every directory has a name, like files. Your *environment* is a collection of items that describe or modify how your computing session will be carried out. It contains things such as where the commands are located and which printer to send your output to. A command or application running on the computer is called a *process*. The sequence of instructions given to the computer from the time you initiate a particular task until it ends it is called a *job*. A job may have one or more processes in it. We will explore each of these elements in a little greater detail later on, but first you need to learn how to initiate a session on a Unix machine.

Logging In and Out

Getting the Login Prompt

Before you can start using the system you must *login* to it. The method that you use to login varies depending on the type of device that you are using to login. Read the section below that is appropriate for you and then read the section, *Entering Your Userid and Password*. In order to login, you **MUST** have an account on the system you are accessing. Remember, different domains require different accounts. You cannot login to Owlnet with a RUF account.

TTY Terminal

If you are using a TTY terminal (a TTY is line-at-a-time oriented as opposed to page oriented) and the screen is blank, you only need to press RETURN and a login prompt should appear on the screen.

Workstation

If the display features a box in the center of the screen with text similar to that in the figure below, then you are using a workstation that is configured to run a windowing system called the X Window system. These machines are called X terminals. (For more information on the X Window system, see the Information Technology document, UNIX 2, *Introduction to the X Window System*.)



If the screen is entirely black, then a screen-saving program is running automatically to protect the monitor from damage. Moving the mouse or pressing the RETURN key should “wake up” the display. (If you see the words “This screen has been locked...” then someone else is using the workstation, but they are temporarily away from their seat. Look for an unoccupied machine.) Move the mouse until the cursor (a black ‘X’) is on top of the white box.

Entering Your Userid and Password

When you applied for your account information, you selected a *userid* and a *password*. This combination of information allows you to access your account. Type your userid using lower-case letters, then press the RETURN key. It is very important that you use *lower-case* letters for your userid. If you make a typing mistake, you can correct it by pressing the DELETE key once for each character you wish to erase. You must make your corrections before you press the RETURN key. If the text you are typing appears in upper-case, see the section, *Troubleshooting*.

After you have entered your userid, the system will prompt you for your password (by displaying the word “Password:” if it is not already on the screen, or by moving the cursor behind the word “Password:” already on your screen). Enter your password and press the RETURN key. Notice that the system does not show or “echo” your password as you type it. This prevents other people from learning your password by looking at your screen.

If you receive a message similar to “Login failed, please try again,” you may have typed your userid or password incorrectly. Try again, making sure to type in your userid and password correctly. If you are still having problems, go to the Consulting Center (713-348-4983, 103 Mudd Labs) and ask for help.

When you have successfully logged on, the system will pause for a moment, and then display a few lines telling you when and from which machine you last logged on, and any messages from the system administrator.

On X terminals, you will get a window containing system information. After reading it, use the left mouse button click on either the “Help” or “Go Away” button, depending on what you want. Help puts you into a help system; Go Away allows you to begin your work.

Your new account is provided with a set of command procedures which are executed each time you log in. You can change part of your UNIX environment by changing these setup files (accounts on Information Technology supported systems are set up to produce a default environment). For further information, check out the Sun manual *SunOS User's Guide: Customizing Your Environment*, available from the Operations Center, 109 Mudd Lab.

The system will then display the *command prompt*. The prompt signals that the system is ready for you to enter your next command. The name of the workstation followed by a percent sign (%) forms the command prompt (e.g. `chub.owl.net.rice.edu%`). Once you finish typing a command, you must always press RETURN to execute it.

Logging Out

Workstations and TTY Terminals

To end a work session, you must explicitly log out of a UNIX session. To do this, type *logout* at the command prompt. Once you have logged out, the system will either display the login prompt again or begin executing a screen saver program.

You should never turn a workstation off. Turning off a terminal does not necessarily log you out. If you are having trouble logging out, see the section, *Troubleshooting*.

X Terminals

To log out of the X Window system from an X terminal, move the cursor into the *console* window (it is labeled “console”), type the command **exit**, and press RETURN. If you try to use the **logout** command in the console window, you will receive the message, “Not in login shell.”

Changing Your Password

You can change your password at any time. We recommend that you change it on a regular basis. At the command prompt, type **passwd**. You will be prompted to enter your old password and be asked twice to enter your new password. Neither your old nor new password will appear on the screen as you type. In order to be accepted, your password must meet the following conditions:

1. It must be at least seven characters long.
2. It must not match anything in your UNIX account information, such as your login name, or an item from your account information data entry.
3. It must not be found in the system’s spelling dictionary unless a character other than the first is capitalized. It must not have three or more consecutively repeated characters or words in the dictionary contained within it.

For example, changing your password from *Kat899* (based on a dictionary word) to *B00z00e* (bad password) will look similar to the following example, except that the keystrokes for you old and new password will not be echoed on the screen.

```
passwd
current password: Kat899
New password (? for help): B00z00e
New password (again): B00z00e
Password changed for userid
```

These are bad examples and will not work, so choose your OWN password. Here is a good technique to follow for creating your password:

```
All she wants to do is dance => Aswtdid
```

On many systems, the password change does not take effect immediately, even though you have finished with the **passwd** command. It can take upwards of an hour for the system to install the new password, due to the scheduling of the password changing process. Thus you should be prepared to use your old password to login again shortly after changing it.

If you should ever forget your password, you can go to the Information Desk in 103 Mudd Lab and request that a new password be generated for you. You will need to bring your Rice ID card with you to identify yourself.

UNIX Commands

The UNIX Shell

Once you are logged in, you are ready to start using UNIX. As mentioned earlier, you interact with the system through a command interpreter program called the *shell*. Most UNIX systems have two different shells, although you will only use one or the other almost all of the time. The shell you will find on Information Technology supported networks is the C shell. It is called the C shell because it has syntax and constructs similar to those in the C programming language. The C shell command prompt often includes the name of the computer that you are using and usually ends with a special character, most often the percent sign (%). Another common shell is the Bourne shell, named for its author. The default prompt for the Bourne shell is the dollar sign (\$). (If the prompt is neither one of these, a quick way to check which shell you are using is to type the C shell command **alias**; if a list appears, then you are using the C shell; if the message, “Command not found” appears, then you are using the Bourne shell). Modified versions of these shells are also available. TC shell (tosh) is C shell with file name completion and command line editing (default prompt: >). The GNU Bourne-Again shell (bash) is basically the Bourne shell with the same features added (default prompt: bash\$).

In addition to processing your command requests, UNIX shells have their own syntax and control constructs. You can use these shell commands to make your processing more efficient, or to automate repetitive tasks. You can even store a sequence of shell commands in a file, called a *shell script*, and run it just like an ordinary program. Writing shell scripts is a topic discussed in the class notes for the *UNIX III-Scripts Short Course*.

About UNIX Commands

UNIX has a wide range of commands that allow you to manipulate not only your files and data, but also your environment. This section explains the general syntax of UNIX commands to get you started.

A UNIX command line consists of the name of the UNIX command followed by its arguments (options, filenames and/or other expressions) and ends with a RETURN. In function, UNIX commands are similar to verbs in English. The option flags act like adverbs by modifying the action of the command, and filenames and expressions act like objects of the verb. The general syntax for a UNIX command is:

command -flag options *file/expression*

The brackets around the flags and options are a shorthand way to indicate that they are often optional, and only need to be invoked when you want to use that option. Also, flags need not always be specified separately, each with their own preceding dash. Many times, the flags can be listed one after the other after a single dash. Some examples later on will illustrate this concept.

You should follow several rules with UNIX commands:

1. UNIX commands are case-sensitive, but most are lowercase.
2. UNIX commands can only be entered at the shell prompt.
3. UNIX command lines must end with a RETURN.
4. UNIX options often begin with a “-” (minus sign).
5. More than one option can be included with many commands.

Redirecting Input and Output

UNIX maintains a couple of conventions regarding where input to a program or command comes from and output from that program or command goes. In UNIX, the *standard input* is normally the keyboard, and the *standard output* is normally the screen. UNIX is very flexible, and it allows you to change or redirect where the input comes from and where the output goes. First, any command that would normally give results on the screen can be directed instead to send the output to a file with the “>” (output redirection) symbol. Thus,

```
date > file
```

directs the system to put the output from the **date** command, which merely reports the time and date as the system knows it, into the file named **file** rather than printing it to your screen. One thing to keep in mind about “>” is that each successive redirection to a particular file will overwrite all of the previously existing data in that file. To append to the end of a file, use “>>” instead. For example,

```
data >> file
```

Another redirection is “<”, which tells the command to take its input from a file rather than from the keyboard. For example, if you have a program that requires data input from the keyboard, you may find that you have to type the same data a large number of times in the debugging stage of program development. If you put that data in a file and direct the command to read it from there you will only have to type the data once, when you make the data file.

```
program < datafile
```

If you do this, you would see the same response from **program** as if you had typed the data in from the keyboard when requested.

You can also combine both kinds of redirection as in,

```
program < datafile > outputfile
```

The data in the file **datafile** will then be used as input for **program** and all output will be stored in **outputfile**.

If you want to accumulate output from different sources in a single file, the symbol “>>” directs output to be appended to the end of a file rather than replacing the previous (if any) contents, which the single “>” redirection will do.

A final I/O redirection is the pipe symbol, “|.” The “|” tells the computer to take the output created by the command to the left of it and use that as the input for the command on the right. For example, we could type:

```
date | program
```

This would use the output of the date command as input to another program.

NOTE: Many, but not all, interactive programs accept input from a file.

Getting On-Line Help with Commands

The standard on-line help facility available with UNIX is electronic reference manuals, known as the man pages, and you access them with the **man** command.

`man command-name`

The man pages provide an in-depth description of *command-name*, with an explanation of its options, examples, and further references. The information is an electronic duplicate of the paper reference manual pages. Use the **man** command for explicit information about how to use a particular command. Use the **-k** option to search for a keyword among the one line descriptions in the help files.

`man -k keyword`

The command **apropos** serves exactly the same function as **man -k** and is used in the same way.

You can read about the **man** command itself using **man**. Type **man man** at the prompt. The UNIX reference manual is divided into eight numbered sections:

1. General User Commands
2. System Calls
3. User-level Library Functions
4. Device Drivers, Protocols
5. File Formats
6. Games (rarely available)
7. Document Preparation
8. System Administration

You can see the command summary for each section by typing:

`man # intro`

where # is one of the eight section numbers.

In addition, other applications that reside on your system may have man pages. These pages can often be called up in the same manner as the operating system man pages.

Case Sensitivity

UNIX is a case sensitive operating system. It treats lower-case characters differently than upper-case characters. For example, the files **readme**, **Readme**, and **README** would be treated as three different files. Most command names and files are entirely in lower-case. Therefore, you should generally plan to type in lower-case for most commands, command line arguments, and option letters.

Special Keys and Control Characters

UNIX recognizes special keys and control-character key strokes and assigns them special functions. A special key such as the DELETE key is usually mapped to the ERASE function, which erases the most recent character that you typed on the current line. A control-keystroke such as CTRL-C is invoked by holding down the key labeled CONTROL and pressing the “c” key (in the same manner that you hold

down the SHIFT key and press the “c” key to generate a capital C). The notation for control characters is usually ^C or CTRL-C. Some standard special keys and control characters are summarized below.

TABLE 1. Special Keys and Control Characters

Special Key	Function/Description
DELETE	Acts as a rubout or erase key. Pressing DELETE once will backup and erase one character, allowing you to correct and retype mistakes.
BACKSPACE	This key is sometimes used as the rubout key instead of the DELETE key. Otherwise, it is mapped as a backspace key, which generates a ^H on the display.
CTRL-U	^U erases the entire command line. It is also called the line kill character.
CTRL-W	^W erases the last word on the command line.
CTRL-S	^S stops the flow of output on the display.
CTRL-Q	^Q resumes the flow of output stopped by CTRL-S.
CTRL-C	^C interrupts a command or process in progress and returns to the command line. This will usually work; if it doesn't, try typing several ^C's in a row. If it still doesn't work, try typing ^\, q (for quit), exit, ^D, or ^Z.
CTRL-Z	^Z suspends a command or process in progress.
CTRL-D	^D generates an end-of-file character. It can be used to terminate input to a program, or to end a session with a shell.
CTRL-\	^\ quits a program and saves an image of the program in a file called <i>core</i> for later debugging.

A Selected Command List

The next few pages summarize many of the basic UNIX commands you need to get started. For further details on UNIX commands not discussed (or that are beyond the scope of this introductory document), consult the system manuals available in the Operations Center, 109 Mudd Lab, and in the labs, or get some practice using the **man** command.

Setup and Status Commands

<i>Command</i>	<i>Purpose</i>
logout	end your UNIX session
passwd	change password by prompting for old and new passwords
stty	set terminal options

date	display or set the date
finger	display information about users
ps	display information about processes
env	display or change current environment
set	C shell command to set shell variables
alias	C shell command to define command abbreviations
history	C shell command to display recent commands

File and Directory Commands

<i>Command</i>	<i>Purpose</i>
cat	concatenate and display file(s)
more	paginator - allows you to browse through a text file
less	more versatile paginator than more
mv	move or rename files
cp	copy files
rm	remove files
ls	list contents of directory
mkdir	make a directory
rmdir	remove a directory
cd	change working directory
pwd	print working directory name
du	summarize disk usage
chmod	change mode (access permissions) of a file or directory
file	determine the type of file
quota -v	displays current disk usage for this account

Editing Tools

<i>Command</i>	<i>Purpose</i>
pico	simple text editor
vi	screen oriented (visual) display editor
diff	show differences between the contents of files
grep	search a file for a pattern
sort	sort and collate lines of a file (only works on one file at a time)
wc	count lines, words, and characters in a file
look	look up specified words in the system dictionary
awk	pattern scanning and processing language
gnuemacs	advanced text editor

Formatting and Printing Commands

<i>Command</i>	<i>Purpose</i>
lpq	view printer queue
lpr	send file to printer queue to be printed
lprm	remove job from printer spooling queue
enscript	converts text files to POSTSCRIPT format for printing
lprloc	locations & names of printers, prices per page
pacinfo	current billing info for this account

Program Controls, Pipes, and Filters

<i>Command</i>	<i>Purpose</i>
CTRL-C	interrupt current process or command
CTRL-D	generate end-of-file character
CTRL-S	stop flow of output to screen
CTRL-Q	resume flow of output to screen
CTRL-Z	suspend current process or command
jobs	lists background jobs
bg	run a current or specified job in the background
fg	bring the current or specified job to the foreground
!!	repeat entire last command line
!\$	repeat last word of last command line
sleep	suspend execution for an interval
kill	terminate a process
nice	run a command at low priority
renice	alter priority of running process
&	run process in background when placed at end of command line
>	redirect the output of a command into a file
>>	redirect and append the output of a command to the end of a file
<	redirect a file to the input of a command
>&	redirect standard output and standard error of a command into a file (C shell only)
	pipe the output of one command into another

Other Tools and Applications

<i>Command</i>	<i>Purpose</i>
pine	electronic mail
bc	desk calculator
man	print UNIX manual page to screen
elm	another electronic mail program

About UNIX Files

Now that you understand UNIX commands, let's discuss the objects manipulated by most commands: files. As we said before, all files have a filename, and UNIX imposes few restrictions on filenames. This makes it easy for you to name your files so that you can easily recognize their contents. You will find it useful to adopt names and classes of names that indicate how important each file is and what connection it has with other files. For example, temporary files used to test commands and options could all begin with a "t." A filename can be up to 256 characters long, consisting of any alphanumeric character on the keyboard except the "/". In general, you should keep your filenames relatively short (to reduce typing effort) and use normal lower-case characters such as letters, numbers, periods and underscores. For instance, if your program calculates employee paychecks, you might call it **payroll**, or if your file is a research paper on Frank Lloyd Wright, you might call it **wright**. Do not include blanks in your filenames as they will make it difficult for you to work with the file. If you do wish to separate letters in a filename, use the underscore ("_") character (as in **wright_paper**) or the hyphen ("-") character. Remember that UNIX is case sensitive, which means it recognizes the difference between upper-case and lower-case letters. For instance, **Wright** and **wright** would refer to two different files.

When you place a single period in the middle of a filename, the part after the period is commonly referred to as an *extension* or *suffix* and usually indicates what type of information is stored in the file. You may use any extension desired; a text file might have the extension **.txt** or **.text**; a note may have the extension **.note**, and so forth. UNIX does not require extensions, but they can be used to help identify similar types of files. Since some UNIX programs (especially compilers) look for certain standard extensions, it is common practice to use the following conventions: **.h** for header files, **.c** for C source files, **.f** for FORTRAN, **.p** for Pascal, and **.s** for assembler source files. So the file **wright.txt** indicates a text file whereas the file **payroll.c** indicates a C program called **payroll**. For more information on programming conventions, see the section, *Additional Resources*.

Some UNIX files begin with a period, for example, **.cshrc** or **.login**. Files that begin with a period will not appear in a normal directory listing and are usually UNIX environment and application setup files.

A large grouping of files and directories is referred to as a *file system*. File systems are related to the disk size and structure, and to the internal structure of UNIX. What you should remember is that users' files and directories are usually on a different file system than the system's files and directories. If the number of users is large, as on OwlNet, the user files and directories may be on more than one file system.

Creating Files

Many files are created using a text editor. A text editor is a program that allows you to enter and save text. You can also use a text editor to manipulate saved text through corrections, deletions, or insertions. The main text editors on Information Technology managed networks are *vi*, *GNU Emacs*, *Pico*, and *aXe*. (Note: *vi* is included with every UNIX system, but GNU Emacs is commonly installed separately by system managers. *aXe* is only available if you are using the X Window system.) You should learn how to use at least one of these tools. Information Technology has tutorial documents on each of these editors. Please see the section, *Additional Resources*, for information on the tutorials.

You can create a file without a text editor by using the **cat** command (short for concatenate) and the ">" (redirect output) symbol. To create a file using the **cat** command, type:

```
cat > new-filename
```

where *new-filename* is the name you wish to give the file. The command **cat** generally reads in a file and displays it to standard output. When there is no filename directly following the command, **cat** treats standard input as a file. The “>” symbol will redirect the output from **cat** into the new filename you specify. **cat** will keep reading and writing each line you type until it encounters an end-of-file character. By typing CTRL-D on a line by itself, you generate an end-of-file character. It will stop when it sees this character. Try it, using this example as a guide:

```
cat > practice
```

When you reach the end of each line, press the RETURN key. You can only correct mistakes on the line you are currently typing. Use the DELETE key to move the cursor back to the mistake and then retype the rest of the line correctly. When you have completed the last line, press RETURN and type CTRL-D.

Displaying Files

Now that you have created a file, you can display it one of several ways. You could use the **cat** command. Just type **cat** followed by the name of the file that you want to see.

```
cat practice
```

Sometimes the files you want to view are very long. When using the **cat** command, the text will scroll by very quickly. You can control the flow of text by using CTRL-S and CTRL-Q. CTRL-S stops the flow of text and CTRL-Q restarts it. If you use CTRL-S, stopping the flow of text, and so on, you must remember to type CTRL-Q or the computer will not display any output, including anything that you type.

more is a program that displays only one screen of information at a time; it waits for you to tell it to continue. Type **more** followed by a filename.

```
more practice
```

The computer will display one screen of text and then wait for you to press the space bar before it displays the next page of text, until you reach the end of the file. Pressing the “?” character will show help for **more**. A utility of greater power called **less** is available on many systems; it allows reverse scrolling of files and other enhancements. It is invoked the same way as **more**.

Listing Files

The **ls** command will list the files in the current directory that do not begin with a period.

Below is a list of options you can tack on to **ls**:

- | | |
|--------------|--|
| ls -a | lists all the contents of the current directory, including files with initial periods, which are not usually listed. |
| ls -l | lists the contents of the current directory in long format, including file permissions, size, and date information. |
| ls -s | lists contents and file sizes in kilobytes of the current directory. |

If you have many files, your directory list might be longer than one screen. You can use the programs `more` or `most` with the “|” (vertical bar or pipe) symbol to pipe the directory list generated as output by the `ls` command into the `more` program. `more` or `less` will display the output from `ls` one page at a time.

```
ls | more
```

Copying Files

To make a copy of a file, use the `cp` (copy) command.

```
cp filename newfilename
```

where *filename* is the file you wish to copy and *newfilename* is the file you are creating.

```
cp practice sample (make a copy of “practice” called “sample”)
```

```
ls
```

```
practice sample
```

The example created a new file called **sample** that has the same contents as **practice**. If **sample** already exists, the `cp` command will overwrite the previous contents. New accounts are often set up so that `cp` will prompt for confirmation before it overwrites an existing file. If your account is not set up in this manner, use the `-i` option (`cp -i`) to get the confirmation prompt, like so:

```
cp -i practice sample
```

Renaming Files

To rename one of your files, use the `mv` (move) command.

```
mv oldfilename newfilename
```

where *oldfilename* is the original filename and *newfilename* is the new filename. For instance, to rename **sample** as **workfile** type:

```
mv sample workfile
```

```
ls
```

```
practice workfile
```

This moves the contents of **sample** into the new file **workfile**. (Note: Moving a file into an existing file overwrites the data in the existing file.) New accounts are often set up so that `mv` will prompt for confirmation before doing this. If your account is not set up in this manner, use the `-i` option (`mv -i`) to get the confirmation prompt.

Deleting Files

To delete files, use the **rm** (remove) command. For instance, to delete **workfile**, type:

```
rm workfile
ls
practice
```

Creating Links Between Files

You can refer to one particular file by different names in different directories. The **ln** command creates a *link*, which “points” to the file. Note that links are simply alternative names for a single file; **ln** does not rename the file (as does **mv**) nor does it make a copy of the file (as does **cp**). It allows you to access the file from multiple directories. Since only one copy of the file actually exists, any changes that you make through one of its links will be reflected when you access it through another of its links, yet if you delete the link, you do not delete what it points to.

Links are useful for cross-referencing files. If you know that you will need to access a file from different directories, creating links is a better alternative to making a copy of the file for each directory (and then having to alter each one every time a change is made to the original). It is also more convenient than having to use the file’s full pathname every time you need to access it. Another use for linking a file is to allow another user access to that particular file without also allowing entry into the directory that actually contains the file. The kind of link you will want to create is called a *symbolic link*. A symbolic link contains the pathname of the file you wish to create a link to. Symbolic links can tie into any file in the file structure; they are not limited to files within a file system. Symbolic links may also refer to directories as well as individual files. To create a symbolic link to a file within the same directory, type:

```
ln -s originalFile linkName
```

where *originalFile* is the file that you want to link to and *linkName* is the link to that file. To create a link in a directory other than that of the original file, type:

```
ln -s originalFile differentDirectoryName/linkName
```

If you create a link within the same directory as the original file, you cannot give it the same name as the original file. There is no restriction on a file’s additional names outside of its own directory. Links do not change anything about a file, no matter what the link is named. If someone makes a link to one of your files, and you then delete that file, that link will no longer point to anything and may cause problems for the other user.

NOTE: You should always use symbolic links when linking to files owned by others!

Important: rm can be very dangerous. Once a file has been removed you cannot get it back, except, possibly, from system backups (which may or may not contain the file). It may take the system administrators several days to recover your deleted file, so use a great deal of caution when deleting files. New accounts are often set up so that rm will prompt for confirmation. If your account is not set up in this manner, use the -i option to get the confirmation prompt.

NOTE: Line printers are used for text-only files. Laser printers are needed to handle graphics or PostScript files. PostScript is a page-description language developed by Adobe Systems, Inc. and was specially designed for creating graphics and typography on a printed page. The option flag **-P** *printername* specifies which laser printer to use and is optional (as indicated by the brackets). When no printer is given, the print command uses the system default printer. However, on some systems such as Owlnet, you must always specify a laser printer with the **-P** flag. For more information on printing commands, use the **man** command to consult the manual pages on **lpq**, **lpr**, and **lprm**.

Printing Files

To print a file, use the **lpr** command:

```
lpr filename or
lpr [-Pprintername] filename (for laser printers only)
```

To get a list of the printers available to your machine, type:

```
lprloc
```

lprloc lists all of the printers that your system knows about, by name, along with their type and location. To get some status information on the printers, use the command **lpstat -p**. Printer accounting information is available by running the command **pacinfo**.

Directories

About UNIX Directories

UNIX directories are similar to regular files; they both have names and both contain information. Directories, however, contain other files and directories. Many of the same rules and commands that apply to files also apply to directories.

All files and directories in the UNIX system are stored in a hierarchical tree structure. Envision it as an upside-down tree, as in the figure below.

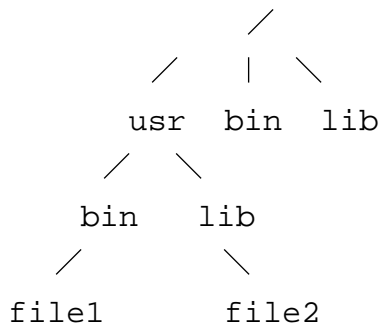


FIGURE 2. UNIX Directory Structure

At the top of the tree is the *root directory*. Its directory name is simply / (a slash character). Below the root directory is a set of major *subdirectories* that usually include **bin**, **dev**, **etc**, **lib**, **pub**, **tmp**, and **usr**. For example, the **/bin** directory is a subdirectory, or “child,” of / (the root directory). The root directory, in this case, is also the *parent* directory of the **bin** directory. Each path leading down, away from the root, ends in a file or directory. Other paths can branch out from directories, but not from files.

Many directories on a UNIX system have traditional names and traditional contents. For example, directories named **bin** contain *binary* files, which are the executable command and application files. A **lib** directory contains library files, which are often collections of routines that can be included in programs by a compiler. **dev** contains device files, which are the software components of terminals, printers, disks, etc. **tmp** directories are for temporary storage, such as when a program creates a file for something and then deletes it when it is done. The **etc** directory is used for miscellaneous administrative files and commands. **pub** is for public files that anyone can use, and **usr** has traditionally been reserved for user directories, but on large systems it usually contains other **bin**, **tmp**, and **lib** directories.

Your *home directory* is the directory that you start out from when you first login. It is the top level directory of your account. Your home directory name is almost always the same as your *userid*.

Every directory and file on the system has a *path* by which it is accessed, starting from the root directory. The path to the directory is called its *pathname*. You can refer to any point in the directory hierarchy in two different ways: using its full (or absolute) pathname or its relative pathname. The full pathname traces the absolute position of a file or directory back to the root directory, using slashes (/) to connect every point in the path. For example, in the figure above, the full pathname of **file2** would be **/usr/bin/file2**. Relative pathnames begin with the *current directory* (also called the working directory, the one you are in). If **/usr** were your current directory, then the relative pathname for **file2** would be **bin/file2**.

If you are using C shell, TC shell, or the Bourne-Again shell, UNIX provides some abbreviations for a few special directories. The character “~” (tilde) refers to your home directory. The home directory of any user (including you, if you want) can be abbreviated from */parent-directories/userid* to *~userid*. Likewise, you can abbreviate */parent-directories/youruserid/file* to *~/file*. The current directory has the abbreviation **.** (period). The parent of the current directory uses **..** (two consecutive periods) as its abbreviation.

Displaying Directories

When you initially log in, the UNIX system places you in your home directory. The **pwd** command will display the full pathname of the current directory you are in.

```
pwd
/home/userid
```

By typing the **ls -a** command, you can see every file and directory in the current directory, regardless of whether it is your home directory. To display the contents of your home directory when it is not your current directory, enter the **ls** command followed by the full pathname of your home directory.

```
ls /home/userid
```

If you are using a shell other than the Bourne shell, instead of typing the full pathname for your directory, you can also use the tilde symbol with the **ls** command to display the contents of your home directory.

```
ls ~
```

To help you distinguish between files and directories in a listing, the **ls** command has a **-F** option, which appends a distinguishing mark to the entry name showing the kind of data it contains: no mark for regular files; “/” for directories; “@” for links; “*” for executable programs:

```
ls -F ~
```

Changing Directories

To change your current directory to another directory in the directory tree, use the **cd** command. For example, to move from your home directory to your **projects** directory, type:

```
cd projects (relative pathname from home directory)
```

or,

```
cd ~/projects (full pathname using ~)
```

or,

```
cd /home/userid/projects (full pathname)
```

Using **pwd** will show you your new current directory.

```
pwd
/home/userid/projects
```

To get back to the parent directory of **projects**, you can use the special “..” directory abbreviation.

```
cd ..
pwd
/home/userid
```

If you get lost, issuing the **cd** command without any arguments will place you in your home directory. It is equivalent to **cd ~**, but also works in the Bourne shell.

Moving Files Between Directories

You can move a file into another directory using the following syntax for the **mv** command:

```
mv source-filename destination-directory
```

For example,

```
mv sample.txt ~/projects
```

moves the file **sample.txt** into the **projects** directory. Since the **mv** command is capable of overwriting files, it would be prudent to use the **-i** option (confirmation prompt). You can also move a file into a another directory and rename it at the same time by merely specifying the new name after the directory path, as follows:

```
mv sample.txt ~/projects/newsample.txt
```

Copying Files to Other Directories

As with the **mv** command, you can copy files to other directories:

```
cp sample.txt ~/projects
```

As with **mv**, the new file will have the same name as the old one unless you change it while copying it.

```
cp sample.txt ~/projects/newsample.txt
```

Renaming Directories

You can rename an existing directory with the **mv** command:

```
mv oldDirectory newDirectory
```

The new directory name must not exist before you use the command. The new directory need not be in the current directory. You can move a directory anywhere within a file system.

Removing Directories

To remove a directory, first be sure that you are in the parent of that directory. Then use the command **rmdir** along with the directory's name. You cannot remove a directory with **rmdir** unless all the files and subdirectories contained in it have been erased. This prevents you from accidentally erasing important subdirectories. You could erase all the files in a directory by first going to that directory (use **cd**) and then using **rm** to remove all the files in that directory. The quickest way to remove a directory and all of its files and subdirectories (and their contents) is to use the **rm -r** (for recursive) command along with the directory's name. For example, to empty and remove your **projects** directory, move to that directory's parent, then type:

```
rm -r projects (remove the directory and its contents)
```

File and Directory Permissions

It is important to protect your UNIX files against accidental (or intentional) removal or alteration by yourself or other users. The UNIX operating system maintains information, known as permissions, for every file and directory on the system. This section describes how to inspect and change these permissions.

UNIX was designed and implemented by computer scientists working on operating system research. Many of the fundamentals of UNIX reflect this origin in academia. A low concern for security is one of the hallmarks of UNIX operating systems. Therefore, unless you act to restrict access to your files, chances are high that other users can read them.

Every file or directory in a UNIX file system has three types of permissions (or protections) that define whether certain actions can be carried out. The permissions are:

read (r)	A user who has <i>read</i> permission for a file may look at its contents or make a copy of it. For a directory, read permission enables a user to find out what files are in that directory.
write (w)	A user who has <i>write</i> permission for a file can alter or remove the contents of that file. For a directory, the user can create and delete files in that directory.
execute (x)	A user who has <i>execute</i> permission for a file can cause the contents of that file to be executed (provided that it is executable). For a directory, execute permission allows a user to change to that directory.

For each file and directory, the read, write, and execute permissions may be set separately for each of the following classes of users:

User (u)	The user who owns the file or directory.
Group (g)	Several users purposely lumped together so that they can share access to each other's files.
Others (o)	The remainder of the authorized users of the system.

The primary command that displays information about files and directories is **ls**. The **-l** option will display the information in a long format. You can get information about a single UNIX file by using **ls -l filename**.

Each file or subdirectory entry in a directory listing obtained with the **-l** option consists of seven fields: permission mode, link count, owner name, group name, file size in bytes, time of last modification, and the filename (the group name appears only if the “**g**” flag is also specified, as in **ls -lg**).

The first 10 characters make up the mode field. If the first character is a “**d**” then the item listed is a directory; if it is a “**-**” then the item is a file; if it is an “**l**” then it is a link to another file. Characters 2 through 4 refer to the owner’s permissions, characters 5 through 7 to the group’s permissions (groups are defined by the system administrator), and the last three to the general public’s permissions. (You can type **id** to verify your userid and group membership.) If a particular permission is set, the appropriate letter appears in the corresponding position; otherwise, a dash indicates that the permission is not given.

The second field in the output from **ls -l** is the number of links to the file. In most cases it is one, but other users may make links to your files, thus increasing the link count. A special warning to people using links to other people’s files: your “copies” of their files can be counted against them by the file quota system available on certain UNIX variants. That is why making links other than symbolic links to other people’s files is strongly discouraged. The third field gives the userid of the owner of the file. The group name follows in the fourth field (if the **-g** option is used in conjunction with **-l**). The next two fields give the size of the file (in bytes) and the date and time at which the file was last modified. The last field gives the name of the file.

```
ls -l myfile
-rw-r--r-- 1 owner 588 Jul 15 14:39 myfile
```

A file’s owner can change any or all of the permissions with the **chmod** (**change mode**) command. The **chmod** command allows you to dictate the type of access permission that you want each file to have. In the previous example the current permissions for **myfile** are read for everybody, write for the owner, and execute by no one.

The arguments supplied to **chmod** are a symbolic specification of the changes required, followed by one or more filenames. The specification consists of whose permissions are to be changed: **u** for user (owner), **g** for group, **o** for others, or some combination thereof (**a** (**all**) has the same effect as **ugo**), how

they are to be changed (+ adds a permission, - removes a permission, and = sets the specified permissions, removing the other ones) and which permission to add or remove (**r** for read, **w** for write, and **x** for execute). For example, to remove all the permissions from **myfile**:

```
chmod a-rwx myfile
ls -l myfile
----- 1 owner  588   Jul 15 14:41 myfile
```

(Note: **chmod a= myfile** achieves the same effect.)

To allow read and write permissions for all users:

```
chmod ugo+rw myfile
ls -l myfile
-rw-rw-rw- 1 owner 588 Jul 15 14:42 myfile
```

To remove write permission for your groups and other users:

```
chmod go-w myfile
ls -l myfile
-rw-r--r-- 1 owner 588 Jul 15 14:42 myfile
```

Finally, to allow only read permission to all users:

```
chmod a=r myfile
ls -l myfile
-r--r--r-- 1 owner 58 Jul 15 14:43 myfile
```

Now the file is protected by allowing only read access; it cannot be written to or executed by anyone, *including you*. Protecting a file against writing by its owner is a safeguard against accidental overwriting, although *not* against accidental deletion.

chmod will also accept a permission setting expressed as a 3-digit octal number. To determine this octal number, you first write a 1 if the permission is to be set and a 0 otherwise. This produces a binary number which can be converted into octal by grouping the digits in threes and replacing each group by the corresponding octal digit according to the table below.

TABLE 2. Symbolic to Octal Conversions

Symbolic	Binary	Octal
---	000	0
--x	001	1
-w-	010	2
-wx	011	3
r--	100	4
r-x	101	5

TABLE 2. Symbolic to Octal Conversions

Symbolic	Binary	Octal
rw-	110	6
rwX	111	7

Thus, if the setting you want is `rw-r--r--`, determine the octal number with the following method:

symbolic	r	w	-	r	-	-	r	-	-
	\	/	\	/	\	/	\	/	\
binary	1	1	0	1	0	0	1	0	0
	\	/	\	/	\	/	\	/	\
octal	6			4			4		

This shows that the octal equivalent of `rw-r--r--` is 644. The following example illustrates that the permissions for **myfile** have been reset to the values with which we began.

```
chmod 644 myfile
ls -l myfile
-rw-r--r-- 1 owner 588 Jul 15 14:44 myfile
```

To change the permissions back to read only, you can execute **chmod** as follows:

```
chmod 444 myfile
ls -l myfile
-r--r--r-- 1 owner 588 Jul 15 14:45 myfile
```

As with files, directories may also have permissions assigned. When listing directories, you may use the **-d** option to keep from descending into the directories you list. Otherwise, the contents of the directories will be displayed as well as their names. Below is an example of permissions assigned to a directory:

```
ls -lgd home
drwxrwxr-x 1 owner caam223 588 Jul 15 9:45 home
```

The directory and the files and directories under it may be read and executed by anyone, but written to only by the owner and users in the `caam223` group. Assuming you are the owner of this directory, you may decide to change the permission to allow only yourself and the `caam223` group to read and execute files in the **home** directory. You would set the permissions accordingly:

```
chmod o-rx home
ls -lgd home
drwxrwx--- 1 owner caam223 588 Jul 15 9:46 home
```

You may decide that only you should be able to alter the contents of the directory. You must remove the write permission for the group.

```
chmod 750 home
ls -l home
drwxr-x--- 1 owner caam223 588 Jul 15 9:48 home
```

An alternative to the previous command is **chmod g-w**.

When you create a file the system gives it a default set of permissions. These are controlled by the system administrator and will vary from installation to installation. If you would like to change the default which is in effect for you, choose your own with the **umask** command. Note that the permission specified by the **umask** setting will be applied to the file, unlike that specified in the **chmod** command, which normally only adds or deletes (few people use the = operator to **chmod**).

First, issue the command without arguments to cause the current settings to be echoed as an octal number:

```
umask
022
```

If you convert these digits to binary, you will obtain a bit pattern of 1's and 0's. A 1 indicates that the corresponding permission is to be turned off, a 0, that it is to be turned on. (Notice that the bit patterns for **chmod** and **umask** are reversed.) Hence, the mask output above is 000010010, which produces a permission setting of `rw-r--` (i.e., write permission is turned off for group and other). Newly created files always have the execution bit turned off.

Suppose you decide that the default setting you prefer is `rw-r-x---`. This corresponds to the masking bit pattern 000010111, so the required mask is 026:

```
umask 26
```

Now, if you create a new file during this session, the permissions assigned to the file will be the ones allowed by the mask value.

Wildcard Characters

Using wildcard characters that allow you to copy, list, move, remove, etc. items with similar names is a great help in manipulating files and directories.

1. The symbol `?` will match any single character in that position in the file name.
2. The symbol `*` will match zero or more characters in the name.
3. Characters enclosed in brackets `[and]` will match any one of the given characters in the given position in the name. A consecutive sequence of characters can be designated by `[char char]`.

Examples of each follow:

1. `?ab2` would match a name that starts with any single character and ends with `ab2`. `?ab?` would match all names that begin and end with any character and have `ab` in between.
2. `ab*` would match all names that start with `ab`, including `ab` itself.
`a*b` would match all names that start with `a` and end with `b`, including `ab`.
3. `s[aqz]` would match `sa`, `sq`, and `sz`.
`s[2-7]` would match `s2`, `s3`, `s4`, `s5`, `s6` and `s7`.

These wildcard symbols help in dealing with groups of files, but you should remember that the instruction:

```
rm *
```

would erase all files in your current directory (although by default, you would be prompted to okay each deletion). The wildcard `*` should be used carefully.

Processes

Every command or program running under UNIX is called a *process*. A sequence of related processes is called a *job*. Your applications and even your shell itself are processes. The windowing system is also a process, or a collection of processes. The UNIX kernel manages the processes on the system, usually without distinguishing among them. UNIX is a multi-tasking system—it allows you to continue to work in the foreground while running one or more jobs in the background. It also runs the processes of many users simultaneously. You could even log off and come back later if the background jobs do not require interaction with you.

Viewing Your Processes

The command **ps** will show you the status of your processes.

```
ps
  PID   TT  STAT   TIME   COMMAND
  4804  p3  S      0:00   -sh (csh)
  1352  p3  R      0:00   ps
  3874  p7  IW     0:25   xclock -g 90x90-0+0
  3875  p7  S      0:48   xbiff -g 90x90-95+0
  3879  p7  S      0:10   twm
  3880  p7  IW     0:00   -bin/csh (csh)
  3892  p9  IW     0:24   /usr/local/bin/elm
```

ps displays the process ID, under PID; the control terminal (if any), under TT; the state of the process, under STAT; the cpu time used by the process so far (including both user and system time), under TIME; and finally, an indication of the COMMAND that is running.

The state of the process is indicated by a sequence of letters. The man pages for **ps** explain what the letters mean if you want to know. For most purposes, you won't really need to know what the letters mean.

Running Background Jobs

Putting a program into an unattended state where it continues to execute is referred to as putting it (the process or job) into the *background*. (Running a program on one machine and displaying its output on another via a windowing system like X is not considered backgrounding the job.)

Adding an & (ampersand) at the end of the command line instructs UNIX to run the job in the background.

```
jobname &
```

The response you receive will be something like this:

```
[1] 5432
```

This particular response means that you have one job running in the background (and its job number is 1), and its process identification number (PID) is 5432. You will need to know the PID if you want to abort the job. This is known as killing a job. To kill the job in the above example, you would type:

```
kill 5432
```

You could also use

```
kill %1
```

or, if there's only one job running called "jobname,"

```
kill %jobname
```

In the C shell, the job number can be used to control which jobs run in background or foreground. The job number is used when switching a job that is processing in the foreground to the background, and one that is processing in the background to the foreground. To do the former, first press CTRL-Z to suspend the job. Then type:

```
bg %jobNumber
```

To switch the job to the foreground, simply type:

```
fg %jobNumber
```

If you have forgotten the job number, type the command **jobs** to see a list of the jobs that are running in the background at the moment.

Note: The rules imposed by system administrators about where and how to run background jobs varies from network to network and changes over time. It is important to stay current with the background job policy of your network.

Process Scheduling Priority

The **nice** command is used to set the processing priority of a command. The priority of a process determines how much attention the system will devote to completing that job. The higher the priority, the more attention a job gets, which implies that it will take less time to complete than the same job run at a lower priority. There are two versions of **nice**. In the C shell, the syntax is:

```
nice -priorityNumber command argument
```

In the Bourne shell, the syntax is:

```
nice +priorityNumber command argument
```

The available priority numbers for users ranges from 1 to 19 with 19 being the lowest priority. In other words, the higher the nice value, the lower the processing priority. (Note: It is important to check the network policy for the required **nice** value for background jobs on your system; they are usually required to be **niced** and your job may be downgraded in priority if it was niced at the wrong value.) Set your command at the required **nice** value or higher. If you do not include a number argument, the value will default to 4 for the C shell and 10 for the Bourne shell.

For example, if you wanted to run a long non-interactive job, and you didn't have to have the results of this job right away, you should run it in the background and set a high **nice** value. Using the C shell, you would type:

```
nice -19 jobname &
```

Remote Login

Sometimes, while you are logged into one workstation, you will find that you would like to be logged in to another workstation, file server, or other UNIX system. The command **rlogin** allows you to do so provided that you have an account on the other system. Type:

```
rlogin newSystem
```

You may then have to supply your password. You should also get the messages about logging in that are used on *newSystem*. If your userid is different on *newSystem* you will have to use the form:

```
rlogin newSystem -l userid
```

Troubleshooting

Logging In

Problem:

The system prints “login incorrect.”

Check to make sure that you are typing your userid and password exactly as they appear on your information sheet (make sure you're using the proper case for all characters). If you still cannot log in, go to the Consulting Center in 103 Mudd Lab and ask for help.

Problem:

The computer types everything in upper-case with slashes in it.

SAMPLE OF/WHAT/PROBLEM LOOKS LIKE

Your userid was entered in uppercase letters, so the system thinks that your terminal can only understand uppercase letters. You need to type logout and then login again making sure that you enter your userid in lowercase letters only. (If the screen displays uppercase as you type your userid, you may

have to turn the CAPS key “off” by pressing it once and try logging in once more.)

Logging Out

If you experience difficulty logging out, check to see if you are having one of the following problems. **Do not turn off the workstation or terminal. This will not disconnect you from UNIX.**

Problem: **The system replies “There are stopped jobs” when you try to log out.**

Type **jobs** to see what they are, then type **kill %%** to terminate all stopped background jobs. After that, try logging out again.

Problem: **The system replies, “Not in login shell.”**

First, type **exit**, press RETURN, and then type **logout** at the prompt, if necessary.

Problem: **Nothing seems to work and you can’t log out.**

You might be stuck in a program or shell other than the login shell. Press RETURN to clear any previous commands; then try typing **CTRL-C, q, :q, :q!, exit, CTRL-D, CTRL-Q, or CTRL-Z** to get back into the login shell. Then try **logout** again. The following sequence might also work if your terminal is NOT connected directly to your real host machine. Press RETURN, then type “~.” followed by RETURN.

If you need additional help, go to the Consulting Center in 103 Mudd Lab during consulting hours or call 713-527-4983. You can also send electronic mail to the Consulting Center by using the address: problem@rice.edu, or better yet, use the web: <http://problem.rice.edu/>

Additional Resources

You now have enough information to get started using UNIX. To learn more about UNIX and specific applications, you can refer to a number of resources. You can use the **man** command to get on-line help. This is the quickest and easiest way to get help during a UNIX session. Virtually every UNIX operation and command has an entry. UNIX documentation written by the vendor for their workstations is available in the document racks of Information Technology maintained labs.

Information Technology has a number of tutorial and reference documents available free of charge outside the Consulting Center in 103 Mudd Lab. Below are some of the UNIX-related documents.

- *Introduction to the X Window System*
- *Customizing the X Window System*
- *Introduction to the vi Editor*
- *Introduction to the aXe Editor*
- *Introduction to GNU Emacs*
- *Compilers, Make, and Debuggers*

Fondren Library has a full set of UNIX reference documentation as well as a large checkout collection of books and manuals on UNIX and other related topics, including some very good introductory books.

The Rice Campus Store has introductory and advanced books on UNIX, as does any bookstore with a computer books section. Your professor may also have some course notes available at the Rice Campus Store.

If you would prefer a hands-on tutorial introduction to UNIX, Information Technology offers free non-credit, one to three hour sessions on various computing topics, including an introduction to UNIX. Short Course calendars are available outside 103 Mudd Lab. For more information, call 713-348-4983 or on RiceInfo (URL: <http://www.rice.edu>).

Problems or Questions

Faculty, Staff, and Graduate Students:

If you have a problem, contact your computing support representative by sending an e-mail message to problem@rice.edu detailing your question. Your query is examined by a staff dispatcher for severity and assigned to the appropriate staff. This is the most effective communication method since computing support staff are often working in the field and unreachable by phone. In addition, the dispatcher is aware of who is on vacation or out ill.

Undergraduates:

If you have a problem, contact your computing support representative by sending an e-mail message to problem@rice.edu detailing your question. Your query is automatically assigned to your College Computing Associate (CCA).

If you need immediate assistance during normal business hours, you can call the Consulting Center at 713.348.4983. During the semester, the Consulting Center has limited evening and weekend hours as well.

To report emergencies, which are urgent system-wide problems (i.e.: all Wiess' network connections are down or all the PCs in a lab are non-functional), contact the Operations Center at 713.348.4989. Staff work 24 hours a day, 365 day a year and can page appropriate administrators for major network or computing problems.

More information is available at <http://www.rice.edu/Computer/student.html>