Quick Study® COMPUTER

UNIX

UNIX INTRODUCTION

The UNIX operating system is a multi-user computing environment designed to provide access to a range of computer applications, tools and utilities to the users of the system. It was designed and developed by Bell Laboratories in the late 60s and 70s for scientists at Bell Labs to help with research, such as writing and publishing technical papers, storing and manipulating data, communicating ideas and working with fellow colleagues.

Universities and colleges obtained UNIX from Bell Labs for research into operating system design. In particular, one university – UC Berkeley – made a variety of improvements and changes to UNIX and released its own version under the Berkeley Software Distribution (BSD) label. Bell Labs also improved its own version of UNIX under the System V name. Each variant provided similar functionality, but BSD always remained a research-oriented tool under the Berkeley umbrella until its last release (BSD 4.4) in the mid-90s. Companies who used UNIX as the primary operating system for their high-end servers and workstations licensed earlier releases of BSD and System V.

The Internet explosion has provided an immense collaborative environment, which has produced a widely popular UNIX clone (in that it was not based on the original UNIX source code) called Linux. Linux provides the same functionality of BSD or SYSV UNIX, but is a completely new design from the ground up which runs on the Intel x86 platforms, as well as other hardware architectures. Other variants of the original BSD UNIX include FreeBSD, OpenBSD and NetBSD.

- **UNIX Applications.** X Window System, Compilers, Mail, Word Processing.
- **UNIX Tools.** Bourne/C/KSH Shell, Utilities (vi, troff, man, lp, who, sort).
- **UNIX Kernel.** Device Drivers, Resource/Process/Memory Management.
- **Hardware.** CPU, Memory, Modem, Hard Drives, Network, Video, I/O.

LOGGING IN/OUT

Logging In

Every UNIX user has a UNIX login username and password. This allows UNIX to keep track of the different people and provide a separate working space for each person within the same system. Users can obtain their UNIX username and password from their system administrator. UNIX is a case-sensitive environment, so take careful note of the upper and lowercase in the username and password.

Once the username and password is obtained, enter them at the login prompt:

```
Login: jsmith
Password: florida
Welcome to SunOS 4.1.4 (GENERIC Kernel)
$
```

The "$" is the UNIX Shell prompt. It may vary, depending on the specific system (the most common variant is "%"). The Shell is where the work will be completed for most of the session.

Changing the Password

After logging in, change the password to something other than what was initially given. It is a good idea to not use common words; instead, use a combination of numbers and letters. A good example is to use the first letters from a verse of a favorite poem, followed by two numerals. For example: *Twinkle, Twinkle, Little Star* becomes *ttls42*.

It is possible to change the password with the **passwd** command:

```
$ passwd
Enter your old password: florida
Enter your new password: ttls42
Reenter your new password: ttls42
Password changed.
$
```

Logging Out

Since UNIX is a multi-user system, it is very important to log out of the session after the work is completed. This releases the terminal back to the login: prompt so that the next user may login. Logging out can be accomplished by typing "exit" at the shell prompt:

```
$ exit
```

Depending on the shell, it is possible to use "logout" or "^D" (***Control-D***).

UNIX SHELLS & COMMAND LINE

Shells provide a command line interface to the UNIX environment and wrap access to files, applications and other tools (thus, the term "shell"). Shells are also known as command interpreters, since they accept commands from the user and interpret them to the UNIX system. The shell allows the user to create simple commands to execute certain tasks or develop complex *scripts*, which can be entire programs within themselves. Depending on the system administrator, the user may have one of the following shells:

- **/bin/sh** - Bourne Shell – the most common shell available on all UNIX systems.
- **/bin/csh** - C Shell – another popular shell designed with shell programming in mind.
- **/bin/ksh** - Korn Shell – an advanced version of the Bourne Shell, which incorporated many of the features of the C Shell.
- **/bin/bash** - Bourne Again Shell – a freely available shell similar to ksh but with more advanced features. Available from http://www.gnu.org.
- **/bin/tcsh** - Tag C Shell – another freely available shell similar to csh. Available from ftp://ftp.gw.com/pub/unix/tcsh.

To find out which shell is in use, type the following command after logging in:

```
$ echo $SHELL
/bin/ksh
$
```

Instructions to run programs are run from the command line prompt of the shell. The command syntax takes the following common form:

command ***[options] [parameters]***

The ***options*** control how the command executes and are usually preceded with the dash sign, "-." The ***parameters*** (also known as arguments) are the data that the command will be processing. These can be numbers, words or even filenames:

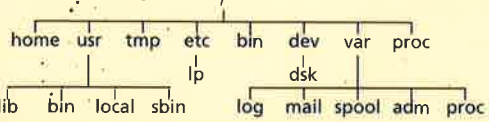
- **ps –ef** - list all the processes running on the current system.
- **wc –l /etc/passwd** - count the number of lines in the file named /etc/passwd.
- **echo $SHELL** - print the value of the SHELL environment variable.

The semi-colon, ";", can be used to specify multiple commands on one line:

ls –l ; pwd ; who

THE UNIX FILESYSTEM

Once logged into the account, the most common tasks will be reading e-mail, editing, running applications, writing and compiling programs, using databases, etc. All of these involve working with files. UNIX uses a hierarchical system to store and retrieve files using what are known as directories. The figure below shows an example of how typical UNIX systems directories are arranged.



The tree-like arrangement starts from the root, "/," directory on downwards. Each branch (i.e., directory) can have sub-branches (sub-directories). Finally, a directory can have individual files (similar to leaves on branches). The forward slash (or divide) sign, "/," is used to delimit each sub-directory. The combination of the directories and files comprise the UNIX file system.

Directories

When logging in to a UNIX system occurs, the user is left in what is known as the HOME directory, which is usually somewhere underneath several levels of sub-directories all starting from the root, "/," directory. With this **pwd** command, the user can tell where the home directory is located.

```
$ pwd
/home/staff/jsmith
$
```

In the previous box, the home directory of the user is the sub-directory named "jsmith," which also happens to be the login user id (this is normal UNIX convention). The "jsmith" sub-directory is underneath the "staff" sub-directory, which is underneath the "home" sub-directory, which itself is underneath the top-most "root" directory. All files that belong to "jsmith" should be created inside /home/staff/jsmith.

Files

A file is a collection of data, such as an email message, information from a database, images, source code, etc. The files reside underneath the user's HOME directory. UNIX allows creation of sub-directories so that the user can arrange the data in an orderly fashion for faster retrieval (it is much easier to have five sub-directories with 10 files in each of them than one sub-directory with 50 files). When referencing files, do so in one of three ways:

- **Simple.** Just the filename, which needs to reside in the current directory.
  Example: *taxes.txt*
- **Absolute.** Specify the entire path to a file starting from the root "/" directory.
  Example: */home/staff/jsmith/finance/taxes.txt*
- **Relative.** Specify the path to a file relative to the current sub-directory.
  Example: *finance/taxes.txt*

Working with Directories

UNIX provides the ability to move swiftly within the UNIX filesystem:

- **pwd** - print the current working directory (CWD).
- **cd** - return to HOME directory.
- **cd *dirname*** - change into the directory named *dir.*
- **cd /** - change into the root directory.
- **cd ..** - change one directory up.
- **cd ../..** - change two directories up from the current directory.
- **cd /usr/local** - change into the directory named /usr/local (absolute path).
- **cd ../tmp** - change into the tmp directory, which exists one level higher.

Creating and removing directories are done with two basic commands:

- **mkdir *dirname*** - create a directory named dirname in the current directory.
- **rmdir *dirname*** - remove the directory named dirname in the current directory. The directory must be empty (i.e., no sub-directories or files within).

Listing Files & Directories

The **ls** command is used to list the contents of directories and the files within them. It can take a variety of options, as shown below:

- **ls** - list contents of the current directory (only shows the names of files and directories).
- **ls –a** - list all files in the current directory, including those that begin with "." (hidden files).
- **ls –b** - force printing of non-printable characters, using octal (base-8) notation. This is useful when there are hidden characters in filenames.
- **ls –F** - append a slash, "/," after directory, an asterisk, "*," after each executable file (including shell scripts) and an at-sign, "@," after each symbolic link.

THE UNIX FILE SYSTEM (*continued*)

- **ls –l** - list contents of current directory in long format (***See:*** *below*).
- **ls –t** - list contents sorted by time-stamp instead of alphabetically.

The long listing (ls –l) is the most common method of invocation of the ls command. It includes the file permission, number of links, owner, group, size and time of last modification for each file listed. For example:

```
$ ls -al
drwx--x--x  10  jsmith  staff  1024     May 24 20:58 .
drwxr-xr-x  13  root    staff  512      May 14 15:54 ..
-rw-r--r--  1   jsmith  staff  2650     Nov  9  1998 .newsrc
-rw-r--r--  1   jsmith  staff  10327    Dec 14 11:05 .pinerc
-rwxr-xr-x  1   jsmith  staff  648      Mar  3 16:04 .profile
-rw-r--r--  1   jsmith  staff  226      Jan  8  1997 .sig
drwxr-x---  2   jsmith  staff  512      May 18 11:33 finance
-rwxr-xr-x  1   jsmith  staff  1320050  Jan  8 14:35 lynx
drwx--x--x  7   jsmith  staff  1024     May  6 21:20 public_html
-rw-r--r--  1   jsmith  staff  5877     May 18 11:34 students.txt
drwxr-x---  2   jsmith  staff  512      May 18 11:33 work
$
```

The first column represents the permissions for each file or directory and is explained in detail below. The second column is the number of links, which is very rarely used in day-to-day sessions. The third column is the owner of the file/directory followed by the group in the fourth column. The fifth column is the file size in bytes. The sixth, seventh, and eighth columns show the time of last modification (month, day, hour and minute, if in the current year or month, day and year for any previous years). The last column is the actual name of the file/directory.

REDIRECTION

The ability to redirect the input and output of an application is one of the most powerful features of the UNIX shells. Using redirection, it is possible to build complex commands by linking the output of one program to the input of another; or, to save the output of one program to a file so that it can be processed later by another program. All UNIX shells provide the following basic redirection capabilities:

Redirection	Meaning
command > *file*	Send the output of *command* to *file* instead of sending it to the screen.
command < *file*	Use the contents of *file* as input to *command* instead of reading from the keyboard.
command >> *file*	Append the output of *command* to *file*.
command1 | *command2*	Send the output of *command1* as input to *command2* (piping).

In addition, each shell provides additional redirection of standard error and standard output:

Bourne/Korn/Bash Shells

Redirection	Meaning
command 1> file1 2> file2	Send the standard output to file1 and standard error to file2 of command.
command 2>&1	Combine the standard error to standard output.
command 2>&1 > file	Same as above, except redirect the combined output to file.

C Shell

Redirection	Meaning
command >& file1	Combine standard error and standard output and send to file1.
command 2>&1 > file1	Same as above, except append to file1 instead of overwriting it.

ACCESS PERMISSIONS

Permissions allow the owner of the file/directory to specify who has access to the file or directory and how he/she can use it. There are three basic types of permissions:

- **Read.** Allows user to view the contents of a file (for example, with an editor). On directories, read access allows you to list the directory contents. Read access is noted with the "r" letter in the permissions column.
- **Write.** Allows user to modify the contents of a file but not delete or rename it. On directories, write access allows the user to create new files and delete or rename existing files. Write access is noted with the "w" letter in the permissions column.
- **Execute.** Allows user to run the file as a program. This requires that the file be either a compiled program or a shell script (if not, unexpected errors will occur). On directories, execute access allows the user to list the contents of the directory. Execute access is noted with the "x" letter in the permissions column.

In addition to the above three permissions, there are a few more permission settings available, but very seldom used. The manual page on chmod(1) describes them in detail.

Ownership

Each UNIX user has a userid (such as "jsmith"), as well as a primary group (such as "staff"). Some newer UNIX systems also allow users to belong to multiple groups (also known as secondary groups). The **id** command can be used to list the userid and groups of the current user.

The first column of an **ls – l** output shows the abbreviated access permission for each file or directory:

```
drwxrwxrwx
```

The first character denotes whether the object is a file, "-," link, "l," or a directory, "d." The next three characters denote the owner access to the file/directory (read/write/execute). The next three characters denote the group access to the file/directory. Finally, the last three characters denote access to the file for everyone else. When trying to access a file, UNIX first checks to see if the user is the owner of the file and uses the owner permissions, if true. If not, it checks if the user belongs to the same group as the file and uses the group permissions, if true. If not, the user is considered "other" and access is based on the permissions set for "other" users.

File permissions can also be represented using numbers:

Permission Number	Literal Meaning
0	None (- - -)
1	Execute (- - x)
2	Write (-w-)
3	Write & Execute (-wx)
4	Read (r- -)
5	Read & Execute (r-x)
6	Read & Write (rw-)
7	Read, Write & Execute (rwx)

Examples:

Permission	Numeric Representation	English Translation
-rw-r--r--	644	owner=read, write group=read other=read
drwxr-xr-x	755	owner=read, write, execute group=read, execute other=read, execute
-rwx------	700	owner=read, write, execute group=none other=none
-r-x--x--x	511	owner=read, execute group=execute other=execute

Setting Permissions

The access permission of files and directories can be changed using the **chmod** command. The read/write/execute permissions can be set or unset, using either the numeric representation or symbolically:

Numeric - chmod ***mode file-list directory-list***

Examples:

```
$ chmod 755 hello.pl
$ ls –l hello.pl
-rwxr-xr-x  1  jsmith  staff  123   May 28 21:08 hello.pl

$ chmod 644 readme.txt
$ ls –l readme.txt
-rw-r--r--  1  jsmith  staff  5571  May 28 21:08 readme.txt

$ chmod 111 final.sh
$ ls –l final.sh
---x--x--x  1  smith   staff  407   May 28 21:08 final.sh

$ chmod 711 public_html
$ ls –ld public_html
drwx--x--x  2  jsmith  staff  512   May 28 21:08 public_html
```

The mode for the numeric parameter to chmod is the absolute numerical permission for the files and/or directories. In order to leave an existing permission alone, a zero is used.

Symbolic - chmod ***who[operation]permission filelist directory-list***

The who parameter can be a letter (or combination of letters), as follows:

Letter	Meaning
u	owner of file (user)
g	group which file belongs to
o	everyone else (others)
a	combination of u, g, o (all)

The operation parameter can be an addition, subtraction or assignment of the permission:

Operation	Meaning
+	Add permission for the specified user or group or others.
-	Delete permission for the specified user or group or others.
=	Set permission for the specified user or group or others.

Finally, the permission itself is a letter (or combination of letters), as follows:

Letter	Meaning
r	Read Permission.
w	Write Permission.
x	Execute Permission.

Examples:

```
$ ls –l
-rwxr-xr-x  1  jsmith  staff  123   May 28 21:08 hello.pl
-rw-r--r--  1  jsmith  staff  5571  May 28 21:08 readme.txt
---x--x--x  1  jsmith  staff  407   May 28 21:08 final.sh
drwx--x--x  2  jsmith  staff  512   May 28 21:08 public_html

$ chmod o-rx hello.pl
$ ls –l hello.pl
-rwxr-x---  1  jsmith  staff  123   May 28 21:08 hello.pl

$ chmod go+w readme.txt
$ ls –l readme.txt
-rw-rw-rw-  1  jsmith  staff  5571  May 28 21:08 readme.txt

$ chmod a+r final.sh
$ ls –l final.sh
-r-xr-xr-x  1  jsmith  staff  407   May 28 21:08 final.sh

$ chmod g=rwx public_html
$ ls –ld public_html
drwxrwx--x  2  jsmith  staff  512   May 28 21:08 public_html
```

WILDCARDS

Wildcards (also known as meta-characters) are special characters, which are used to specify existing filenames. When used in the command line, the UNIX shell expands the wildcard characters and tries to match them to filenames. Once matched, the names are then passed to the program specified.

Wildcard	Meaning
?	Match any single character in filename.
*	Match any number of characters (zero or more).
[]	Match any of the characters enclosed within the brackets.

The most common use of wildcards is for listing, moving or deleting files.

Examples:

```
$ ls -F
find.sh   kernel.c      readme.txt rfc3.txt   tcp3.c
hello.c   public_html/  rfc1.txt   tcp1.c     tcp4.c
hello.pl  pwd.c         rfc2.txt   tcp2.c
$ ls *.c
hello.c kernel.c pwd.c tcp1.c tcp2.c tcp3.c tcp4.c
$ ls ???.c
pwd.c
$ ls *[12].*
rfc1.txt  rfc2.txt        tcp1.c      tcp2.c
$ ls *[1-3].c
tcp1.c    tcp2.c          tcp3.c
$ ls r*
readme.txt    rfc1.txt    rfc2.txt    rfc3.txt
$ ls rfc[!12].c
hello.c   kernel.c        pwd.c       tcp3.c    tcp4.c
$ rm *.c
$ ls
find.sh   public_html     rfc1.txt    rfc3.txt
hello.pl  readme.txt      rfc2.txt
$ mkdir tmp
$ mv *.txt tmp
$ ls -F
find.sh*     hello.pl*    public_html/   tmp/
$ ls tmp
readme.txt  rfc1.txt     rfc2.txt     rfc3.txt
```

PRINTING

UNIX provides the ability to send files to printers, which may physically be connected to the UNIX system or via the LAN. Depending on the type of UNIX being used, there are two different ways to submit and control print jobs. Check with the system administrator to find out which one is available, as well as the names of the printer queues.

Printing on BSD-Based UNIX

Command	Meaning
lpr –P printer file	Send file to the printer queue named printer.
lpq –P printer	Check the print jobs on the print queue named printer.
lprm –P printer job-id	Delete the print job job-id from the print queue named printer.

Printing on System V-Based UNIX

Command	Meaning
lp –d printer file	Send file to the printer queue named printer.
lpstat	Check the print jobs on the print queue named printer.
cancel printer job-id	Delete the print job job-id from the print queue named printer.

In addition to the parameters shown above, the printing programs accept a wide variety of additional parameters that can be used to control how jobs are printed, such as page width, page title, fonts used and number of copies sent. Check the manual page for the commands and the system administrator for any specific printer. In addition, the **pr** command can be used to pre-format text files in a wide variety of ways before sending them to the printer.

FINDING FILES & DIRECTORIES

The **find** and **grep** utilities can be very useful when trying to locate a file or search for a file based on the contents of the file. If the name of the file in question is known (but not its location), the find utility can be used to search the directories for its location:

```
$ find . –name resume.txt –print
./work/personal/resume.txt
$
```

Although the find utility is extremely powerful, its most common use is of the following form:

find *directory-name* –name *filename* –print

where directory-name is the starting directory to begin the search and filename is the name of the file being located.

Wildcards are allowed for filename but if used, they must be enclosed within single or double quotes. The most common starting directory is your HOME directory since it is very rare that you would save your personal files in any location, except underneath your login directory. Find also accepts certain parameters, which affect what and how it searches:

- **-type [dlf]** - search for file, which is a **d**irectory or **l**ink or regular **f**ile.
- **-mtime n** - search for files modified at **n** days (or more) ago. If **n** is a negative number, then the search is for files modified less than **n** days ago.
- **-ls** - used instead of –print. List the found files using output similar to **ls –l**.

Examples:

```
$ find $HOME –name '*.c' –print
./work/kernel.c
./work/testing.c
./login.c
$ find $HOME –name 'rfc*.txt' –print
./text/rfc1.txt
./text/rfc2.txt
$ find . –type d –ls
119681   1  drwx—x—x  7  jsmith  staff
  1024  May 28 14:10 ./work
108282   1  drwxr-xr-x  6  jsmith  staff
   512  Apr 28 09:27 ./text
$
```

If what the user is searching for is not based on filenames but rather on the contents of the file, the grep utility can be used. The format of grep is:

grep –ilnv *search-pattern file-list*

where *search-pattern* is the word being searched for and *file-list* is the list of files on which to perform the search. *Search-pattern* can be a simple word (or set of words if enclosed within double-quotes) or a regular expression. By default, only the lines that match are printed; however, this can be changed with the following parameters:

- **–i** - search for *search-pattern* but ignore the case.
- **–l** - print the names of the files, which have *search-pattern* in them.
- **–n** - precede each line with its line number.
- **–v** - print all lines, *except* those that contain *search-pattern*.

Examples:

```
$ grep John students.txt
John Smith
John Sawer
$ grep –i mcdonald students.txt
Mary McDonald
Duane MCDonald
Lisa mcdonald
$
```

ENVIRONMENT VARIABLES

UNIX shell uses environment variables to set up certain default settings for the login session. One example is the SHELL variable, which contains the name of the UNIX shell you are currently using. Here is a list of the most common variables used by the different SHELLs.

Bourne/Korn/Bash

- **HOME** - user's login home directory
- **PATH** - colon-separated list of directories used by the shell when searching for programs to run when typed on the command line
- **MAIL** - user's mail filename, which is used to check for new mail
- **MAILPATH** - colon-separated list of filenames used to check for new mail
- **MAILCHECK** - how often to check for new mail
- **PS1** - user's shell prompt displayed on the command line
- **PS2** - secondary shell prompt when waiting for more input from the user
- **CDPATH** - colon-separated list of directories used by the cd command when searching, for the directory specified to change into
- **SHELL** - name of the current user's shell
- **DISPLAY** - specify the display name for X Window Applications
- **$$** - process ID of the current shell
- **$!** - process ID of the most recent background command
- **$?** - return code from the previously executed command

C Shell (note the case)

- **home** - user's login home directory
- **path** - space-separated list of directories used by the shell when searching for programs to run when typed on the command line
- **cwd** - current working directory of the shell
- **history** - size of command history buffer
- **prompt** - user's shell prompt displayed on the command line
- **cdpath** - space-separated list of directories used by the cd command when searching for the directory specified to change into
- **DISPLAY** - specify the display name for X Window Applications
- **shell** - name of the current user's shell
- **$$** - process ID of the current shell
- **$status** - return code from the previously executed command

Viewing Shell Variables

The **echo** command can be used to view the current value of shell variables. The variable name must be preceded with the dollar sign, "$."

```
$ echo $HOME
/home/staff/jsmith
$ echo $PATH
/bin:/usr/sbin:/usr/etc:/usr/ucb:/usr/local
  /bin
$ echo $PS1
$
```

Setting Shell Variables

The values of shell variables can be changed to suit the user's environment. The most common variables changed are the search path (PATH) and prompt (PS1 or prompt). The **export** command is used to export variable names to new sub-shells. The user can also create his/her own variables by setting them to an initial value.

Bourne/Korn/Bash

```
$  PATH=/bin:/usr/sbin:/usr/etc:/usr/ucb:
  /usr/X11R6/bin
$ PS1='Command> '
Command> MYNAME="John Smith"
Command> echo $MYNAME
John Smith
Command> export PATH PS1
```

The C Shell provides the **set** and **setenv** commands to set environment-variable values:

```
$ set path=(/bin /usr/sbin /usr/etc /usr/ucb
  /usr/X11R6/bin)
$ set prompt='Command> '
Command> setenv MYNAME "John Smith"
Command> echo $MYNAME
John Smith
Command>
```

QUICK TIP The autologout environment variable in csh and tcsh can be set to logout the current shell after a given number of minutes of inactivity.

ELECTRONIC MAIL

UNIX provides users with the ability to communicate via electronic mail (email). The most common mail reader programs are listed below:

- **mail** - very simple command-line mode mailer available on most systems.
- **mailx** - similar to mail but with some more advanced features to support mailboxes.
- **elm** - full-screen-based mail reader.
- **pine** - very popular, full-screen-based mail reader, which also supports file attachments using MIME.

ARCHIVING FILES & DIRECTORIES

It is often useful to be able to archive a set of files and/or directories into one file so that they can be transported as one unit to a different system and later extracted. This is similar to the ZIP utility on PC/Windows systems. Unix provides this facility using some command line utilities:

Creating Archives

tar –cvf archive.tar *[files/directories]*	Archive the specified files and/or directories into a file named archive.tar.
compress archive.tar	Compress the file named archive.tar and rename it to archive.tar.Z.
gzip archive.tar	A better compression program (not available on all UNIX systems). File is renamed to archive.tar.gz.

Extracting Archives

uncompress archive.tar.Z	Uncompress the file named archive.tar.Z and rename it to archive.tar.
gunzip archive.tar.gz	Uncompress the file named archive.tar.gz and rename it to archive.tar (not available on all UNIX systems).
tar –xvf archive.tar	Extract the contents (both files and directories) of archive.tar into the current directory.

The **tar** utility is used to store the contents of files and directories specified on its command line into a single file. It will automatically recurse into sub-directories within the directories specified, and store all the sub-directories and their files in the archive file. Thus, it is a very efficient way of backing up all the files into one file, which can then be copied to another system for transport or safekeeping. The **compress** and **gzip** utilities are used in conjunction with tar to compress the archive file to save space. Although not necessary in the archiving process, they can often compress files to almost half their original size.

REMOTE CONNECTIONS

The ability to login to a remote UNIX system via a LAN or the Internet is provided by two primary applications: **telnet** and **rlogin.**

telnet hostname – establish a remote session with *hostname*.

This is the most common way of logging into a remote system. Once a connection is established, a login: prompt is displayed, and the remainder of the session resembles a session as if the remote system was the user's local system. Depending on the distance and network speeds, the user might notice delays in typing and seeing results. When the remote session ends, the telnet program exits and returns to the shell prompt on the local system. If for some reason you need to disconnect without logging off the remote system, the user can type ***Control-]***, which returns to a *telnet>* prompt from which the user can type ***quit*** to disconnect the session and return to the local shell.

rlogin hostname - establish a remote session with *hostname*.

Rlogin works in a similar fashion to telnet, except some additional information is passed along the connection (such as the username). By doing this, the remote session can skip straight to the password prompt, from which point the session is the same as a telnet login. To disconnect without logging off, type **~.**, which will return the user to the local shell. Rlogin also has facilities to allow configuration rlogin to bypass the password checks using configuration files. Although this can make the remote session easier to login to, care must be taken to avoid opening accounts to logins from any user.

Both rlogin and telnet transmit the text of a session over the network un-encrypted. A fairly recent equivalent to telnet/rlogin is **ssh,** which uses public-key encryption to authenticate and transmit data between the two systems in a secure fashion. Check with the system administrator for its availability on the UNIX system.

JOB CONTROL

The C, Korn and Bash shells provide the user with the ability to run programs in the background. This allows the user to multi-task within one login window by controlling programs (jobs) using shell commands. A program can be run in the background by appending the ampersand "&" after the command is typed:

```
$ sort students.txt > sorted.txt &
[1] 700
$ jobs
[1]+ Running    sort students.txt > sorted.txt &
$ ls -F
students.txt  sorted.txt  public_html/  Mail/
[1]+ Done    sort    students.txt > sorted.txt &
$
```

The above example runs the sort command (which sorts the contents of *student.txt* and sends the sorted output to *sorted.txt*) in the background using the "&." Immediately after the return key is entered, the shell returns the job number, "[1]," and the process id of the job, "700." Currently running background jobs can be listed with the **jobs** command. Once a job is complete, the shell will inform the user by displaying the job number and command line.

To make a currently running command (also known as foreground process) switch to running in the background, you can type ***Control-Z***. This will suspend the current program and return you to the shell prompt. Typing **bg** will unsuspend the process and run it in the background. The **fg** command can be used to bring a background process into the foreground. Both commands allow an optional job number parameter so that you can specify which job to move to the background or bring to the foreground. ***Notes:*** *A background process will automatically be suspended if it is waiting for user input. Any output from a background process will be printed to the screen if the standard output/standard input is not redirected to a file. All unfinished background jobs are killed when a user logs out. To prevent this, use the* ***nohup*** *command.*

Job Control Command Summary

Command	Meaning
command &	Run the command in the background.
fg %#	Move the background job number identified by # to the foreground.
bg %#	Move the job number identified by # to the background and continue running.
jobs	List the current background jobs.

FILE TRANSFER PROTOCOL (FTP)

The ftp program allows the transfer of files between systems that are connected over a network. The two systems can be connected across the Internet or via a Local Area Network (LAN). Using ftp requires an account on the server side, although some public access systems on the Internet allow anonymous logins.

Unix provides the ftp program via the command-line:

ftp *hostname* - connect to the system name specified in *hostname* in order to initiate file transfer.

Once ftp is able to connect to *hostname*, it will be necessary to provide a username and password. The system administrator of *hostname* will provide this information. An example ftp connection is shown below (the text typed by the user is in bold):

```
$ ftp ftp.university.edu
Connected to ftp.university.edu.
220 to ftp.university.edu FTP server
  (Version wu-2.4.2-academ [Wed
  May 21 14:02:26 EDT 1997]) ready.
Name to ftp.university.edu:jsmith):
  jsmith
331 Password required for jsmith.
Password: florida
230-
230-Hello jsmith@company.com,
230-
230-The current time is Sat May 22
  21:17:08 1999.
230-You are user 2 out of 100.
230-
230-If you have any problems, please
  email sysadm@university.edu.
230-
230-Thank You.
230-
230 User jsmith logged in.
ftp>
```

If establishing an anonymous ftp connection, it is possible to login as the user named "anonymous." The password will be the user's full email address. Anonymous FTP sites are usually provided by many companies to allow individuals to download software and drivers for the products of the company. In addition, some major shareware software sites on the Internet also allow anonymous FTP connections.

Once logged in, the transfer of files from the host computer can be initiated (i.e., downloading) or to the host computer (i.e., uploading). In order to do so, ftp provides a set of commands, which can be used:

- **ascii** - switch to ASCII transfer mode. This mode is used when transferring files which are purely text.
- **binary** - switch to binary transfer mode. This mode is used when transferring binary data or application programs between the systems.
- **bye** - terminate the ftp session and return to the shell prompt (command-line).
- **cd *directory_name*** - change into the directory specified by *directory_name* on the remote system.
- **delete *remote-file*** - delete the file specified by remote-file on the remote system.
- **dir *specification*** - list the contents of the current directory on the remote system. The list can be limited by providing a regular expression filter in *specification*. For example:
  dir *.tar - list only files ending with .tar.
  dir *.[ch] - list only files ending with .c or .h.
  dir a*z - list only files beginning with "a" and ending with "z."
- **get *remote-filename local-filename*** - download a file from the host computer to the local computer. The file will reside on the same directory where the ftp command is started. If *local-filename* is omitted, the name of the file will be the same as *remote-filename*. To change the destination directory, see the **lcd** command.
- **hash** - show a hash mark, "#," for every 1,024 bytes transferred (system-dependent).
- **help** - provide a list of commands supported by the ftp program.
- **lcd *directory-name*** - change into the directory specified by *directory-name* on the local system. This allows transferring of files from/into different directories on the local system without having to quit the ftp, change into a different directory and/or restart ftp.
- **ls** - Similar to the **dir** command.
- **mdelete *remote-filelist*** - delete the files specified in *remote-filelist* on the remote system.
- **mget *remote-filelist*** - download multiple files from the remote system, as specified in *filelist*. For example:
  mget *.c - download all files ending with .c.
  mget t1.zip t2.zip h.c - download the three files named t1.zip, t2.zip and h.c.
  mget *.zip *.h - download all files ending with .zip and .h.
- **mkdir *directory-name*** - create a directory named *directory-name* on the remote system.
- **mput *filelist*** - upload multiple files to the remote system from the current directory on the local system, as specified in *filelist*, which is similar to the examples for mget.
- **prompt** - turn off the confirmation prompting for each file when using mget or mput.
- **put *local-filename remote-filename*** - upload the file named *local-filename* to the remote site under the name specified in *remote-filename*. If *remote-filename* was omitted, the remote name will be *local-filename*.
- **pwd** - print the current working directory of the remote system.
- **recv** - same as **get**.
- **rmdir *directory-name*** - delete the directory named *directory-name* on the remote system.
- **send** - same as **put**.
- **status** - show the current status of ftp.
- **quit** - same as **bye**.

UNIX provides a variety of text editors, which can be used to write programs, edit letters, prepare manuscripts or email. The following is a list of the most commonly available editors:

- **ed -** a line editor available on all UNIX environments.
- **vi -** a full-screen editor, popular among UNIX die-hards. Harder to learn but very functional once mastered.
- **emacs -** a multi-purpose editor, which can be used to do almost all facets of development within its environment.
- **joe -** a simple-to-use, full-screen editor.
- **pico -** another simple-to-use, full-screen editor, which is also used by the PINE email program.
- **xedit -** a simple X-window system editor.
- **nedit -** another X-window system editor very similar to *Notepad* under *Windows*.

The VI Editor

The **vi** editor is a full-screen editor available on all UNIX systems. Although it may look hard to learn because of the cryptic commands, once mastered, **vi** is not only a powerful editor, but can also be used to do a variety of text manipulations. The **vi** editor, unlike most other editors, has two modes:

Input mode – add text to the file.

Command mode – tell the editor to perform certain tasks, such as delete characters, words or lines, copy lines to a buffer, repeat operations and save the file.

The *Escape* key is used to switch from input mode to command mode. To switch back to input mode, there are a variety of commands, as shown below.

VI Command Summary

General Commands	Meaning
Escape	Switch from input mode to command mode.
Control-C	Stop the currently running command.
Control-L	Redraw screen.
:q	Quit the editor (if there are unsaved changes, the editor will warn the user).
:q!	Force quit (discard any unsaved changes).
:w *filename*	Write the current file to disk. If *filename* is specified, then save the text to it instead.
:w! *filename*	Force-write the current file to *filename*.
:wq	Save the current file and quit the editor.
:n	Go to the next file specified on the command line.
:n!	Switch to the next file specified on the command line and discard any changes to the current file.
:set showmode	Show the current mode to the user (useful for beginners).
:set tabstop=#	Set the tab expansion to # characters.
!!command	Run command and read its output into the current text.
:r *filename*	Read-in the contents of *filename* into the current file.

Switch to Input Mode Commands	Meaning
a	Append text after the current cursor position.
A	Append text at the end of the current line.
i	Insert text at the current cursor position.
I	Insert text at the beginning of the current line.
o	Insert text on a new line after the current line.
O	Insert text on a new line before the current line.

Although some implementations of **vi** allow the use of the arrow keys and the PGUP and PGDN keys, movement in **vi** is always available with the following command mode letters:

Movement Commands	Meaning
j *(or Enter)*	Move the cursor down one line.
k	Move the cursor up one line.
l *(or Space)*	Move the cursor forward one character.
h *(or Backspace)*	Move the cursor back one character.
0	Move cursor to beginning of line.
^	Move cursor to first non-blank character in line.
$	Move cursor to end of line.
G	Move cursor to end of file.
w	Move cursor forward one word.
b	Move cursor back one word.
)	Move cursor forward one sentence.
(	Move cursor back one sentence.
}	Move cursor forward one paragraph.
{	Move cursor back one paragraph.
H	Move cursor to top of current screen.
L	Move cursor to bottom of current screen.
M	Move cursor to middle of current screen.
Control-F	Move forward one screen.
Control-B	Move backward one screen.
Control-E	Move screen forward one line (leave cursor on same line).
Control-Y	Move screen back one line (leave cursor on same line).

** These commands can be preceded with a number N so that the command will automatically be repeated N times.*

Modification Commands	Meaning
dd	Delete the current line.
x	Delete the character under the cursor.
r	Replace the character under the cursor with the next character typed.
R	Overwrite the remainder of the line (switches to input mode).
u	Undo the last command (only one level of undo).
yy	Yank current line to internal copy buffer.
p	Paste internal buffer after the current line.
P	Paste the internal buffer before the current line.

Search/Replace Commands	Meaning
/*string*	Search forward (search string is typed next).
?*string*	Search backward (search string is typed next).
n	Repeat previous search (moving forward or backward).
f	Search the current line for the character typed next.
F	Search backward in the current line for the character typed next.
t	Same as "f" but the cursor is positioned on the character before.
T	Same as "F" but the character is positioned on the character after.
/regexp	Same as / but regexp is a regular expression.
?regexp	Same as ? but regexp is a regular expression.
:s/*original*/*replace*/	Replace first occurrence of *original* with *replace* on current line.
:s/*original*/*replace*/g	Replace all occurrences of *original* with *replace* on current line.
:%s/*original*/*replace*/	Replace first occurrence of *original* with *replace* on every line.
:%s/*original*/*replace*/g	Replace every occurrence of *original* with *replace* on every line (same as global search and replace).

The EMACS Editor

Emacs is a publicly available editor developed at MIT. It has an internal language, based on LISP, which allows the editor to be extended in order to add new functionality. Emacs is language-sensitive in that it will enable certain features when it recognizes the file type that is being edited. For example, if a C source file is loaded, Emacs will switch to C-mode, which will automatically format the code as it is typed. Emacs also has the ability to edit and view multiple files (buffers) at the same time.

Movement Commands	Meaning
Control-f	Move forward one character.
Control-b	Move backward one character.
Esc-f	Move forward one word.
Esc-b	Move backward one word.
Control-e	Move to end of line.
Control-a	Move to beginning of line.
Control-n	Move down by a few lines.
Control-p	Move up by a few lines.
Esc-e	Move to the end of the sentence.
Esc-a	Move to the beginning of the sentence.
Esc-{	Move to the beginning of the paragraph.
Esc-}	Move to the end of the paragraph.
Esc-<	Move to the beginning of the file.
Esc->	Move to the end of the file.
Control-v	Move forward one screen.
Esc-v	Move backward one screen.
Control-l	Redraw screen.

Modification Commands	Meaning
Control-d	Delete the character under the cursor.
Delete	Delete the word to the right of the cursor.
Esc-c	Capitalize word.
Esc-u	Uppercase word.
Esc-l	Lowercase word.
Esc-d	Delete the characters from the start of the line to the cursor.
Control-k	Delete the characters from the cursor to the end of the line.
Control-Space	Mark current position as beginning of region.
Control-w	Delete region from mark to cursor position.
Control-y	Paste most recent, deleted text to current cursor position.
Esc-w	Copy region to buffer.

Search/Replace Commands	Meaning
Control-s	Search forward (search string is typed next).
Control-r	Search backward (search string is typed next).
Esc-x replace-string	Prompts for original and replacement string and does a global replace.
Esc-x query-replace	Same as above, but prompts users before each replace.

COMMAND SUMMARY

- **banner *text*** - print out *text* in large format text.
- **bc** - basic calculator.
- **cal *[month] [year]*** - print the calendar for the current year. A specific month in the current year or a specific year (using all 4 digits) can be specified on the command line.
- **cat –n *[filelist]*** - print the contents of the files specified in *filelist* to the screen.
  –n - precede each line with a line number.
- **clear** - clear the screen and leave the cursor at the top of the screen. On some systems, this can be mimicked by typing ***Control-L*** at the command line.
- **cp –ip *[source_file] [target_file]*** - copy the contents of *source_file* into *target_file*.
  –i - prompt user for confirmation if *target_file* already exists.
  –p - preserve ownership and permission of *source_file* into *target_file*.
- **cp –r *[source_dir] [target_dir]*** - recursively copy the contents (files and subdirectories) of *source_dir* into *target_dir*.
- **date** - print the current date and time.
- **du –sk** - summarize the disk usage for the current directory and its subdirectories and print the result for each subdirectory.
  –s - total all the subdirectories and print the result as one number.
  –k - print the output in kilobytes (1,024-byte blocks instead of 512-byte blocks).
- **echo –n *[string]*** - print the words in string to the screen.
  –n - do not print a new line after printing the *string*. Leave cursor on the same line (On some systems, this is done by appending a "\c" at the end of *string*.).
- **calendar** - calendar utility.
- **compress –v *[filelist]*** - compress the contents of the files specified in *filelist* by using a mathematical algorithm. Compressed files are appended with the .Z extension. Text files are usually compressed 50% to 60% of their original size. ***Also see uncompress and gzip.***
  –v - print out the results of the percentage reduction.
- **finger *user[@hostname]*** - display information about local or remote users. If *@hostname* is included, finger tries to establish a connection to the computer named *hostname* and prints the information on *user* on that *hostname*.
- **ftp** - File-Transfer-Protocol is used to transfer files between UNIX and other systems. ***See the section on FTP for details.***
- **gzip *[filelist]*** - compress the contents of the files specified in *filelist*, using a better compression algorithm than **compress**. Compressed files are appended with the .gz extension. Text files are usually compressed 60% to 70% of their original size. ***Also see compress and gunzip.***
- **gunzip *[filelist]*** - uncompress the files specified in *filelist* (all of which must have the .gz extension) which were previously compressed with gzip. The uncompressed files will be created without the .gz extension. ***Also see uncompress and gzip.***
- **head -# *[filelist]*** - print the first 10 lines of the files specified in *filelist* to the screen.
  –# - specify the number of lines to print (***Ex: head –50 users.txt*** will print the first 50 lines of users.txt). ***Also see tail.***
- **id** - return the user and group names and the user and group ids of the user who invoked the command.
- **lynx** - text-mode Web browser.
- **mesg –ny** - turn off/on the ability to accept **write** messages from other users.
  –y - accept write requests (can also use just "y").
  –n - refuse write requests (can also use just "n").
- **more *filename*** - page through *filename*. If *filename* is omitted, more reads from standard input.
- **mkdir –p *[directory_list]*** - create the directories named in *directory_list*.
  –p - create all non-existent parent directories for *directory_list*.
- **mv –fi *[source] [target]*** - move the file or directory specified in *source* to the file or directory specified in *target*. By specifying a filename for *source* and *target*, **mv** works as a rename command. If target is a directory, the file or directory specified in *source* is moved under *target*.
  –f - do not prompt the user if *target* already exists.
  –i - prompt the user whenever the move would overwrite an existing *target*.
- **nohup** - run a command immune to shell exits.
- **ping *hostname*** - send a test connection network packet to verify that the remote *hostname* is reachable over the network.
- **rm –if [filelist]** - delete the files specified in *filelist*.
  –i - prompt the user for confirmation before deleting each file.
  –f - delete all specified files, ignoring file permissions and without confirming with the user.

Note: –i and –f are mutually exclusive (use only one or the other).

- **rmdir *[directory_list]*** - remove the directories specified in *directory_list*.
- **rm –rf *[filelist] [directory_list]*** - recursively remove the directories and their sub-directories and files specified in *filelist* and *directory_list*.
  –r - specify the recursive deletion of the directories (required).
  –f - delete all specified files and directories, ignoring file permission and without confirming with the user.
- **sleep *[time]*** - suspend execution for the period specified in *time* (in seconds).
- **talk** - talk to another user.
- **tail ±# *[filename]*** - print the bottom 10 lines of the file named *filename*.
  +# - start printing after skipping #-1 lines from the top of the file ***(Ex: tail +20** will print the contents of users.txt after skipping the first 19 lines).*
  –# - start printing the bottom # lines of the file ***(Ex: tail –5 users.txt** will print the bottom 5 lines of users.txt).*
- **telnet** - connect to a remote (usually UNIX-based) system across a network connection. ***See the section on remote connections for details.***
- **uncompress –v *[file]*** - uncompress the files specified in *filelist* (all of which must have the .Z extension) which were previously compressed with compress. The uncompressed files will be created without the .Z extension. ***Also see compress, gzip and zcat.***
  –v - print out the results of the percent expansion.
- **uniq** - remove duplicated lines from a file.
- **w –hsu** - display information on the users who are currently logged in.
  –h - suppress the header information.
  –s - display the output in short format.
  –u - display only the header information which contains the current time, number of users logged in and system load averages.
- **wc –cwl *[filelist]*** - count the number of characters, words and lines of the files in *filelist* and print the results to the screen.
  –c - only count the number of bytes/characters.
  –w - only count the number of words.
  –l - only count the number of lines.
- **who [am I]** - display a list of users who are currently logged into the system. ***Also see w.***
  am i - display just the information on the user invoking who.
- **write *username*** - write a message to *username* who must also be logged onto the current system. ***Also see mesg.***
- **zcat [filelist]** - uncompress the contents of the files specified in *filelist* (all of which must have the .Z extension), which were previously compressed with compress, to the screen instead of to a file.

LOGIN & LOGOUT SHELL SCRIPTS

Each UNIX shell has a script that is automatically run when a user logs in. In addition, some shells also run a script as the user is logging out. The following is a list of the scripts for each shell and their purpose. All the scripts must be located in the home directory for each user.

Bourne/Korn/Bash Shell

Script Name	Purpose
.profile	Automatically executed when the user logs in. Used to setup the PATH, prompt and other environmental settings.
.kshrc	Korn Shell: Executed for every instance of ksh.
.bashrc	Bash Shell: Executed for every instance of bash.
/etc/profile	System-wide login script setup by administrator for global settings.

C Shell

Script Name	Purpose
.login	Automatically executed when the user logs in. Used to setup the PATH, prompt and other environmental settings.
.cshrc	Executed for every instance of csh.
.logout	Executed when the user is logging out.
/etc/login	System-wide login script setup by administrator for global settings.

QUICK TIP

The UNIX system provides manual pages on almost all the available commands. The "**man**" utility is used to display the manual page for a certain command. For example: "man ls" will display the manual page for ls. "man -k keyword" will search the manual pages for the "keyword" and show a one-line summary of each match.

CREDITS
Author: Mahesh Neelakanta
Design: Michael D. Adam

PRICE
U.S. $5.95

Screen representations may vary depending on the version of the software that is installed. This guide is based on the software version that was shipping at the time of publication and is accurate to that version. For specific changes to a software application, see the Read-Me file provided with the software application.

ISBN-13: 978-1572223974-4
ISBN-10: 1572223974-9
50595
9 781572 223974
6 54614 20397 7