Introduction to Unix from http://cnl.web.arizona.edu

Index

Related Resources

Operating Systems

An Operating System (e.g., Windows 95, Windows 98 etc. from Microsoft; Unix systems like Linux, Irix, and Solaris) organizes your files and talks to the hardware. This means the OS translates from the very crude organization system that really exists on your hard drive to the hierarchical system that it presents to you.

Essentially, it works like this: The Hard Drive has an area set aside that holds a table of addresses or "pointers" to blocks of information on the rest of the HD. The table's content and organization, the number and size of blocks on the HD etc. are dependent upon the particular OS. Although Unix systems are not exactly the same as each other, they are very similar to one another; just as Windows 95 and Windows 98 are very similar to one another.

Files (A Comparison of Windows and Unix)

Under Microsoft Windows, file names consist of two parts: a main part and an extension, separated by a special character, the dot (e.g., bird.jpg, fred.xls). The extension tells Windows which program to use to open the file (e.g., *.xls files are opened by MS Excel). For some popular extensions (*.jpg, *.gif), when you install new programs, they will "steal" the association by registering themselves as the default application for opening that type of file. To view or modify the file associations on Microsoft Windows, open an explorer window, and choose View-->Folder options, then click the tab that says "file types").

Under Unix everything is a file (directories are files too). Unix systems do not use file extensions to associate files with programs. In fact, you can use multiple dots in the names of your files (e.g., fred.tar.gz) or directories (e.g., rc2.d). The only time the dot has special meaning under Unix is when it begins a file name: all hidden files begin with a "." (e.g., .cshrc).

Windows retains case information in file and directory names, but it does not use case to distinguish between files (If you have a file named Fred.txt, and you create a new file in the same directory called FrEd.TXT, Windows will overwrite the first one with the new one)

Under Unix, upper and lower case are functionally different (You could have the following files in one directory, and they would be treated as distinct by the operating system: fred.txt, Fred.txt, FrEd.TXT etc.).

In the rest of this introduction I adhere to 3 simple conventions for teaching you commands:

- > = shell prompt (You don't type this, it simply stands in for whatever the system displays in the command window when it is ready for you to type something.)
- The rest of the bolded characters are what you actually type.
- An explanation in () may follow, but it will not be bolded and you should not type it.

To display a list of files at the shell prompt:

>ls (Think of this as standing for "list")

To display a list including hidden files (those that begin with a dot):

>ls -a (Think of this as standing for "List all")

To display maximal information about the file types and content:

>file *

Wild Cards: If you want a command to work on multiple files, then you need to know about wildcards (metacharacters). You may be familiar with the *. The * matchs zero or more characters:

Thus *.txt = all files in this directory that end with the characters .txt

P10201*=all files in this directory that begin with P10201 followed either by nothing or by any other number of characters.

Other metacharacters are:

? matches one and only one character (e.g. P1??01 would match any file that started P1 followed by any two charcters followed by 01.)

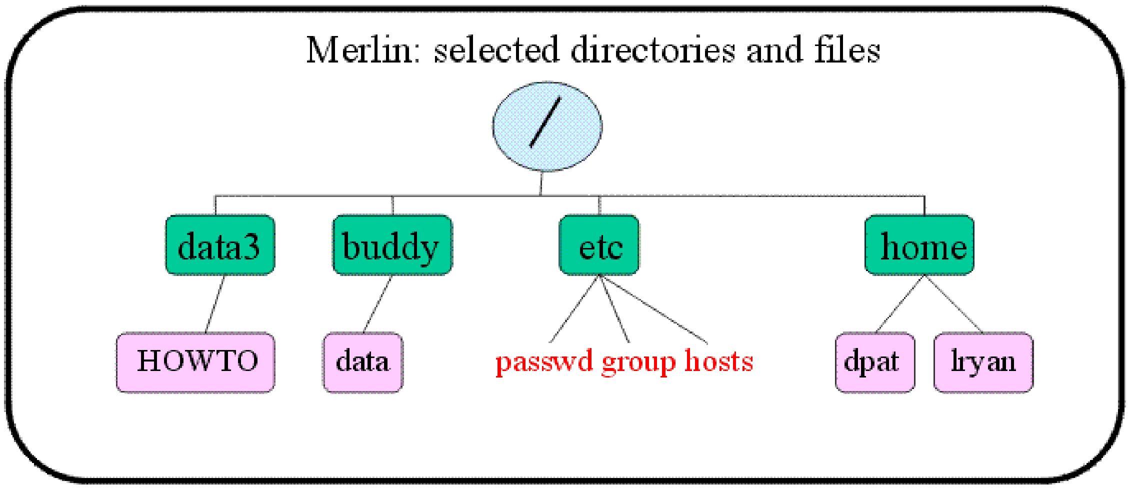
[abc] matches any one character in the bracket. (e.g. joe[abc] would match joea, joeb, joec, but not joed or joeab; joe[abc]? would match joeab)

[a-d] matches any one character in the range abcd. Note that the "range" is defined by the sequence of characters in the ascii character chart. All lower case letters are separate from the upper case letters in the ascii chart.

Other wildcards exist, but are not so useful as these.

Unix File System

In order to move around the file system, you should think of it as an inverted tree:



The tree above displays several selected directories and files on merlin. At the highest level is root "/". Below root, in green, are several important directories:

/data3 is the directory corresponding to the raid tower on merlin. It is permanantly mounted on merlin. It can be automounted from the other machines as /merlin/data3/. However, it is important to recognize that if you automount /merlin/data3/ from another machine, you are still technically on that machine, not merlin.

/buddy is a special directory. It is a mountpoint for the automounted raid tower /data which is permanantly mounted on buddy (hence the directory name). If you are logged on to merlin and you automount /buddy/data/, you are still on merlin NOT buddy. Automounting a raid tower or hard drive does not change which machine you are on, it just gives you access to that storage area. If you want to be on a different machine, you must login to the machine, either by sitting at the terminal and logging in or by using ssh to contact and login to the other machine. To automount /data from merlin, simply type:

>cd /buddy/data (Think of "cd" as standing for "change directory")

All unix systems have /etc, a special directory for system administrators. Understanding some of what it contains will help you understand a lot about how unix works. The three files listed in red in the tree diagram are particularly interesting:

"passwd" is a file containing all the "user" (programs can also be users) names and ID numbers. User ID numbers identify each user to the system and are used to control access to files and directories. Each user has an ID number, a group ID number, a home directory and a default shell specified. Nowadays, passwords are actually stored in a different file called the "shadow" file.

Sample line: fred:x:60111:10:Fred Faker:/home/fred:/bin/tcsh

"group" is a file that contains the names and ids of all groups on the system. Again, some groups are for programs, not people, and the system identifies groups by their ID numbers, not their names. Most users will belong to the group "user". But root (superuser) belongs to the group "sys". Some users may belong to special groups that correspond to individual projects, (e.g., "catprod" is such a group on our core machines). Some users may belong to more than one group. Like the user ID, group IDs are used to control access to files and directories, either by granting special access to group members, or by denying access to those who are not group members.

Sample line: user::10:

"hosts" is a file that lists the ip addresses of a select group of machines and the names that the system is to treat as equivalent to those ip addresses. By including short names for machines that one frequently needs to contact, processes like telnet, ftp, ssh and scp can be made easier (i.e., one can say ssh merlin, rather than ssh merlin.psych.arizona.edu or ssh 128.196.98.209 from any machine that lists merlin as equivalent to 128.196.98.209)

Sample line: 128.196.98.209 merlin merlin.psych.arizona.edu

/home is the directory on most unix machines that contains the home directories of individual users. From anywhere on the system, you can go home by typing the command to change directory to home:

>cd ~ (~= twiddle)

The Path

You can find out where you are in the system by typing the command to print working directory:

>pwd

Every directory contains two hidden directories by default:

"." and ".."

A single dot refers to the directory itself "here". Two dots refer to the parent directory in the tree (e.g., in our tree above / is the parent of the home directory; home is the parent of the lryan directory; / is the grandparent of lryan, etc.)

To go up one directory in the tree (i.e., to go to the parent directory):

>cd ..

To copy a file from /home/joe/public_html/HOWTO to the current working directory (i.e., "here", wherever you are when you type the command):

>cp /home/joe/public_html/HOWTO .

To navigate through the tree, you specify a "path".

An absolute path starts with / because it specifies the instructions for getting to the target, starting at the top of the tree and specifying every intermediate directory from root "/" to the target. Intermediate directories are separated by "/" in unix. Such paths can be quite long, but they can get you to the destination from anywhere in the system:

>cd /home/joe/public_html/HOWTO/

This tells the system: Go to the root directory. From root, find your child "home". Go there. From "home", you will find that one of your children is "joe". Go there. From "joe", you will find that one of your children is "public_html". Go there. From "public_html", you will find that one of your children is "HOWTO". Go there and stop.

A relative path does not start with "/". Relative paths have the advantage of being short, but the disadvantage of being dependent on the current starting place:

If I am in /home/joe/ and I want to go to HOWTO, I can type:

>cd public_html/HOWTO/

From /home/joe, I can also specify a relative path to a sibling directory or any child of that sibling:

>cd ../mary/public_html/

If I want to, I can use a relative path to move up the tree to the top

>cd ../../ (This would take me up two levels in the tree)

>**whereis ls** (This asks the system to tell me the absolute path to the ls command.) >which ls (This asks the system to tell me which version of the ls command I am using, often, this simply means the system will tell me the absolute path to the ls command.)

Some paths are specified in your shell configuration file (like the paths to basic commands). To find out which paths are specified there, type:

```
>echo $PATH
```

The paths specified in the shell configuration file can be altered. Read more about the shell below.

The Shell

Experienced unix users become attached to particular shells. A shell is a command window under unix. Machines may default to providing you with a bourne shell, bash shell (bourne-again shell), csh (c-shell), tcsh (t-shell), or any one of several other shells. Most of us use the tcsh because it has command completion and history features that are extrememly handy. Each user's shell customization file is a hidden file in their home directory.

>ls -la (in your home directory this will show you all hidden files, including your .cshrc; Think of it as standing for "list -long -all").

The tcsh uses the .cshrc as its customization file, unless a .tcshrc is available in the same directory. The .cshrc contains information about the paths to look for commands, the way the prompt looks, shortcuts called "aliases", and environment variables. The .cshrc is a text file that you can examine or edit. Similar "rc" files exist for the other shells: .bashrc for the bash shell, etc. Sometimes, there is an additional profile for the shell that sets up default paths and hides in the /etc directory. You might wish to examine the aliases, for example, and add new ones.

After you make changes to your .cshrc, they don't apply until you source the file

>source .cshrc (typed from your home directory, this will refresh the shell and apply your new changes. Keep in mind that your changes will apply only in the shell window you actually typed in or in any new windows you open. If you have other windows open at the time you source the .cshrc, they will not be refreshed.).

If you login to a machine and want to know which shell you are in

```
>echo $SHELL
```

To verify that tcsh exists on the machine

```
>which tcsh
```

If you want to start a tcsh (providing that it exists on the machine)

```
>tcsh
```

Your system administrator can alter the passwd file for you to make tcsh (or any other shell on the machine) your default shell (if you like it and it is not your default).

Ownership and Permissions in Unix

Ownership

Under Unix, every file is owned by a user and a group. To see the user and group to whom a file belongs:

```
>ls -l
```

Below is a sample result of ls -l. "silly" belongs to the user "joe" and the group "user".

```
-rw-r--r-- 1 joe user 0 Apr 7 02:37 silly
```

The purpose of user and group ownership is to control permissions on files and directories: whether you can read , write (change, delete, rename), or execute (run a program).

Superuser: Every system has one user called "superuser" or "root" who can do anything. This user can become any other user, read and alter any files and directories, change passwords, add and remove users etc.

To change ownership of a file or directory if you are not superuser, you must be the current owner. So, for "silly", joe can give the file to another user, thus:

>chown lryan silly

To change the group that owns "silly" from "user" to "catprod":

>chgrp catprod silly

Who am I? You may sometimes operate as different users at the same time (e.g., You might want to be yourself in one window and ryanlab in another window).

>su joe (this switches you to being joe...you must know the ryanlab password)

>exit (if you switched to being joe, and now you want to go back to being yourself)

>whoami (tells you who you are at the moment)

Groups: You may belong to more than one group. If this is true, there will be a default group you belong to when you login, and one or more secondary groups.

>id (Tells you which groups you belong to)

>id -a (Tells you what group you are in at the moment)

>newgrp catprod (changes your group from the current group to "catprod" if you belong to that group)

Permissions

These appear as a sequence of 10 characters: -rw-r--r--.

The first character tells you what kind of file it is (e.g., - = file, d=directory, l=link).
The next 3 characters are the ordered read, write and execute permissions for the user who owns the file:
read permission (- =no, r=yes, read permission),
write permission (- =no, w=yes,write permission),
execute permission (- =no, x=yes, execute permission);
Next, the 3 ordered permissions for the group,
Finally, the 3 ordered permissions for others, (everyone who is not that user or a member of that group).
-rw-r--r-- 1 joe user 0 Apr 7 02:37 silly

In the above example, the first character indicates that "silly" is the default "-" (a basic file, not a link, directory or something else).
joe can read and write to "silly", but does not have permission to execute silly.
Members of the group "user" can read "silly", but do not have permission to write to or execute it.

Others can read "silly", but cannot write to or execute silly.
Directory permissions have interesting (and not immediately obvious) consequences:

You need read permissions on a directory to list its contents.
You need write permissions on a directory to create, copy or remove files and directories in it.
You need execute permission on a directory to cd to it.
Changing permissions: chmod (Change Mode?) is used to change permissions. This can be done in many ways, but here is a straightforward one. Remember that:

Permissions are r (read), w (write), and x (execute).
Permissions can be added (+) or taken away (-).
Permissions apply to u (user), g (group), and o (other).
Examples:

>chmod u+x silly (Adds execute permissions to the file "silly" for the user, "joe" in the above example)

>chmod go-r silly (Removes group and other read permissions from the file "silly")

>chmod -R g+w Stupid (Adds group write permissions recursively to a directory "Stupid" and all files and subdirectories in it.)

Copy, Move and Remove Files and Directories

Files

>cp silly funny (Copy the file "silly" to the file "funny", resulting in two copies of the file with different names)

>cp silly /home/joe/public_html/ (Copy the file "silly" to the directory /home/joe/public_html/, resulting in a copy in the original location and in the specified directory, both named "silly")

>cp silly /home/joe/public_html/crazy (Copy the file "silly" to the directory /home/joe/public_html/ and name it "crazy", resulting in a copy in the original location named "silly"and a copy in the specified directory, named "crazy")

>mv silly funny (Move the file "silly" to the file "funny", resulting in only one file, "funny". Note that mv is functioing like some sort of rename command in this case. It is ALWAYS more dangerous to move something under unix than it is to copy it, because there is no undo function in the shell.)

>mv silly /home/joe/public_html/ (Move the file "silly" to the specified directory)

>rm silly (Remove the file "silly")

>rm *.MR (Remove all files in the current directory that end with .MR)

Directories

>mkdir Stupid (Make a directory in the current location named "Stupid")

>rmdir Stupid (Remove the empty directory "Stupid" from the current location. This only works if the directory is empty. Remember that hidden files in "Stupid" could prevent you from removing "Stupid")

>rm -r Stupid (Remove the directory "Stupid" and all of its contents recursively. rm -r is a VERY dangerous command. Most systems will prevent this command from applying to the hidden directory .. since this would start deleting files from the parent directory of "Stupid"...and then the grandparent of "Stupid" etc. etc ad infinitum. Systems are less likely to protect you from the consequences of removing a linked file. A linked file is like a shortcut to a file somewhere else in the system. Recursive commands can thus jump from the current directory to somewhere else via a linked file. It is always better to take a long time and check carefully what you are deleting, rather than going for the quick option. Your home area and the various system areas (bin, etc, var ...) might well have linked files. Your data directories on the Raid towers usually won't.

>rm -rf Stupid (Like the above command, but the "f" forces the removal without asking you about each file)

>cp -R Stupid Nuts (Copy the directory "Stupid" and all of its contents to the directory "Nuts", resulting in two complete copies of the directory and its contents.)

>mv Stupid Nuts (Move the directory"Stupid" to the directory "Nuts". This is like renaming the directory if Nuts does not already exist. You end up with one directory called "Nuts" which contains all the content of the original "Stupid" directory.

However, if Nuts does exist, then Stupid (and all of its contents) are moved into Nuts.

Archiving and Zipping

Because the gzip utility is intended to be used for files, if you wish to zip up a directory, you must first archive (tar) it to turn it into a file. If you want to zip up specific files in a bunch of directories and subdirectories, click here learn more about the find command:

>tar cvf - Packages > NewPackages.tar (Tar the directory "Packages" to create the file "NewPackages.tar"; c=create, v=verbose, f=file)

>gzip NewPackages.tar (Zip up NewPackages.tar to reduce its size. This will create the file NewPackages.tar.gz)

>gzip -r temp (Zip up all files under temp, recursively looking through subdirectories for files to zip).

Now if you want to unzip and untar Packages:

>gunzip NewPackages.tar.gz (This unzips "NewPackages.tar.gz" and leaves you with the file "NewPackages.tar" instead.

>tar xvf NewPackages.tar (This untars "NewPackages.tar" and creates the directory "Packages" which contains all the original files and subdirectories. It also leaves the original "NewPackages.tar", so now you have both. x=extract, v=verbose, f=file)

>gzip bird.tiff (gzip a file)

>gunzip bird.tiff.gz (unzip a zipped up file)

Viewing and Editing a Text File

>cat silly (To view a short text file named "silly")

>nedit silly (On the sgis this brings up the file "silly" in a notepad-like editor. If you have permission, you can make and save changes to the file. Note that this ties up your shell prompt so you have to either finish and exit nedit to use the shell, or open another shell)

>nedit silly & (On the sgis: like the above command, but the "&" tells the system to run the process [i.e., nedit] in the "background", so it doesn't tie up the shell prompt. You can open new files from the nedit menu without starting new copies on nedit.)

>less silly.txt A viewer, much easier to use and more flexible than "more". Less is the default viewer for man pages on buddy, holly, merlin and charlie. You can view binary files with less, without fear of damaging them. You can use less to search files for words and you can switch into vi from less by typing "v". Less can be used over a telnet or ssh connection.

>less --help or **>man less** will show you options. **h** while in less will bring up the help screen.

Starting less: less <filename> (e.g., **>less cheatsheet.txt**)
q (for quit)

Moving:
- Scroll up and down using the arrows (this depends on the machine.
- j and k can be used to scroll as in vi if the arrows don't work).
- Spacebar takes you down one screen, enter down a line (as in "more").
- g takes you to the first line in the file, G takes you to the last
line in the file.

Other:
- **v** (while in the viewer, puts you into vi)
- **/word** (while in the viewer, a forward slash followed by a word or regular expression will search for the first occurrence of the word and will highlight it. n will find the next instance etc. See various
flags for the search, /word* looks through multiple files, but you need to choose Ctrl-n to switch to the next file, and n to search for the next occurrence. ESCAPE-u will turn off highlighting from

the current search.
- less can be used to view or search multiple files e.g. >less *.txt

On the sun, right click on the background, go to "Applications" on the popup menu, and choose text editor.

Size Matters

Sometimes you will want to know how big a file (or a directory) is. This gets confusing, because the numbers are huge...like counting Lira. In addition, systems are not always clear about what units they are presenting size information in. For example, many Linux systems default to bytes, but allow you to choose Kilobytes or megabytes. Irix allows you to choose kilobytes but not megabytes. This table shows you how the different units of measurement are related. Below are discussions of how big different media and applications are. Also, crucial commands and what they return are described.

Bit	Byte	Kilobyte	Megabyte	Gigabyte	Terabyte
2 states	8 bits	8,000 Bits	8,000,000 Bits	8,000,000,000 Bits	8,000,000,000,000 Bits
	1 Byte	1,000 Bytes	1,000,000 Bytes	1,000,000,000 Bytes	1,000,000,000,000 Bytes
		1 Kb	1000 Kb	1,000,000 Kb	1,000,000,000 Kb
			1 Mb	1000 Mb	1,000,000,000 Mb
				1 Gb	1000 Gb
					1 Terabyte

In the above table, all items in a column are equivalent:

A bit = 2 states, positive and negative.
A Byte = 8 bits.
A Kilobyte (Kb) = 8,000 bits or 1000 bytes.
A Megabyte (Mb) = 8 million bits or 1 million bytes or 1000 Kilobytes.
A Gigabyte (Gb)= 8 billion bits or 1 billion bytes or 1 million Kilobytes or 1000 Megabytes.
A Terabyte = 8 trillion bits, 1 trillion bytes, 1 billion kilobytes, 1 million Megabytes or 1000 Gigabytes.
Beginning with bytes, each level is 1000 times as large as the previous level (A byte is only 8 times as big as a bit, BUT a Kilobyte is 1000 bytes, a Megabyte is 1000 Kilobytes etc.)

Storage Media
To give you some perspective:

A floppy disk holds 1.44 Mb
A zip disk may hold 100 Mb or 250 Mb
A CD holds about 650 Mb (but some are bigger)
Most of us have hard drives that are between 10 and 40 Gb.
The RAID towers hold about 200 Gb each.
All 3 CNL raid towers + whatever hard drive space is actually on our sgis amounts to over 1/2 a terabyte of data storage (~600 Gb or 600,000 Mb).

Stuff you Store
So how big are the things we actually store around here?

A typical exam (a set of raw MR files and some P-files) is about 200 Mb, so it easily fits on a CD.
An MR file is ~ 136 Kb

A P-file might be between 20 and 60 Mb.
When you list files on Unix, "ls -l" represents the size in bytes.

Applications vary considerably in size, but big ones are sometimes up around 200 Mb.

Other examples (obtained by right clicking a folder under Windows 98 and choosing "Properties"):

Matlab: Almost 170 Mb
Norton Antivirus: 15 Mb
Office 97: Almost 100 Mb
SPSS: 85 Mb
WinZip: 2 Mb

Figuring it Out
On Unix:
>du -ks Fred
This will give you a Disk Usage Summary in Kilobytes of the space used by the directory Fred.

>ls -l
This will give you the size in bytes of all files. Directory sizes do NOT reflect the contents of the directory.

>df -k
This will give you the free (and used) space on the devices (partitions) on the system in Kilobytes.

On Windows:
The size of files is displayed in the Explorer.
The size of directories (folders) can be obtained by right clicking and selecting "Properties".
The size (and free space) on a partition can by found by right clicking on the drive letter and choosing "Properties".

Unix Help

You have already been introduced to many of the important commands you will need for navigating the file system and understanding permissions. Below are some hints for finding help and a review of several crucial commands not already covered.

Opening a Shell: Sometimes you may have a graphical user interface available (where you can drag files with a mouse), but you will always have a shell or "window" available where you can type commands. On the sgis, you can open a new shell window by going to the Toolchest (upper left corner of screen) and clicking on "Desktop", then choose "Open Unix Shell" from the popup menu. On other systems, you can try right clicking on the background and looking through the options on the popup menu, or click likely looking icons on the desktop. On the sun, try right clicking on the background, then choose "Tools" and "Terminal" from the popups. See "History" and "Filename Completion" below to learn more about shell features.

Help for Commands: All unix systems have man (manual) pages for the standard commands:

>man ls (This will display the manual page for "ls" using a standard viewer like "more". Hit the SPACE bar to scroll down a screen at a time and "q" to quit)

Man pages can be frustrating to read, and they are probably not the best way to learn a command, but they are always available.

>man man (to learn more about man pages).

For commands that require arguments (e.g. cp requires an argument, because you cannot copy unless you specify the file to be copied and where it will be copied), you can type the name of the command without any arguments and hit ENTER to display a brief usage message. Try this for many of the imaging routines (especially MGH routines like inorm):

>inorm

We have two useful books in the dataroom for general unix commands:

Ray, DS and EJ Ray. (1998). Visual QuickStart Guide: Unix. Peach Pit press.
Gilly et al., (1992). Unix in a Nutshell. O'Reilly & Associates.

Peachpit Press books are great learning tools. O'Reilly makes a whole line of excellent computer books, but they assume a bit more knowledge than the Peachpit Press Visual QuickStart books. Unix in a Nutshell is an alphabetical reference to commands.

Check out the "help" in the toolchest on the sgis. There is a similar reference facility available on suns.

Tips and Tricks

Cut and Paste
You can use the left mouse button to highlight text in the shell and then move anywhere else (another shell, an nedit window etc.) and click the middle mouse button to paste the text at the current cursor location.

Filename Completion
This is another feature of t-shell. If you begin to type the name of a file and press TAB, the shell will complete the name of the file for you. If there is more than one file that starts with a similar sequence (i.e., study1_reg_norm+orig.BRIK and study1_reg_norm+orig.HEAD), and you type st TAB, the shell will complete as much of the name as it can (study1_reg_norm+orig.) and then beep at you to give it more input. In this example, if you type a B and TAB it will complete the name of the file.

Find Command
The find command is useful, but its syntax is difficult. Here we illustrate several examples of the find command.

One great way to reduce the size of a directory without tarring it up, is to gzip all the files individually. Of course, this can take a long time if you do it manually, so here is a useful find command that will gzip everything in the current directory and below (make sure you are in the right place before you run it).

First, you might wish to do a "fake" run of the command, to find out what it will do (use the -ok flag instead of the -exec flag)

So here is the fake run version:

```
>find . \! -name '*.img' -ok gzip -v {} \;
```

And here is the real version:

```
>find . \! -name '*.img' -exec gzip -v {} \;
```

Note that this won't zip up directories, just the files inside directories. It'll warn you that it can't zip up directories (that's fine). These commands will find all files here (.) and below that are not (\!) *.img files and gzip them up. You could remove the "not" and find and zip all img files:

```
>find . -name '*.img' -exec gzip -v {} \;
```

Here's another example, to give you some idea of the power of find:

```
>find . -group catprod
```

This command finds and lists all files and directories, here and below, belonging to the group catprod.

History
Your default shell on the sgis is set to remember the last 100 commands you entered at the shell prompt. Use the up and down arrows to scroll through the commands. You can also edit the

commands after you find them and before you run them, just use the left and right arrows to move through the command, then insert and delete all you want. Note that this is a setting on our sgis that you won't find set up on every unix machine. If you don't seem to have it, you may want to ask your system administrator about tcsh (t-shell) and whether you can access it.

Clients and Servers

You should understand the difference between a machine that serves up services (like ssh, scp, telnet, and ftp) and a machine that is simply a client for such services.

Servers: Most unix machines (unlike most Windows PCs) come with server capability. When the machine starts, the server daemons start. Each daemon (e.g., telnetd, sshd, ftpd etc.) listens for incoming calls on its port. A file called inetd.conf in /etc lists all the daemons that the system administrator allows to listen (the system administrator may choose to turn these daemons off if they represent a security threat).

Most machines can act as **clients**. The client program is like a telephone that only allows you to make outgoing calls, but not receive them. So our PCs all have telnet and ftp clients built in, and we've added ssh and scp clients. Because our PCs do not run servers for these services, we cannot sit at home or at a unix station and ftp, telnet, ssh or scp to the PCs.

Let's consider an example: The telnet client can telnet to a remote machine...but it does not listen for telnet requests from remote machines. If your machine has a telnet server and a permanant IP address (think of this as its telephone number)...then it can receive an incoming telnet request (it has a daemon listening on the telnet port). Otherwise, it simply acts as a telnet client, allowing only outgoing telnet requests.

In the case of our CNL machines, very few daemons are running. In particular, telnet and ftp daemons are turned off (they are not allowed to listen). However, our machines are still clients for such services...so you can telnet to a remote machine from buddy, but no remote machine can telnet to buddy.