This document was orginally developed by the Campus Computing Organization at Caltech, edited locally to provide a better match to the computing environment at our site. (this editing is in progress, 12/94).

Unix Tutorial

Contents

- Some Common and Useful Unix Commands For Files

A Sample Login Session

The School of Oceanography has several Unix workstations available for use in two public workrooms, 265 MSB and 212 ORB (more information). Any of these accept logins over the campus network as well. Our principal server is tsunami.ocean whose environment is mirrored on reef, shoal, dune and jetty.

Logging On

When you first connect to one of the Unix computers you will see the prompt:

```
    login:
```

If you see only the prompt `Password:` you probably used `rlogin`. `rlogin` assumes that your username is the same on all computers and enters it for you. If your username is different, don’t worry, just press <CR> until you see the `login:` prompt and start from scratch.

At the `login:` prompt, type in your username. *Be careful to type only lowercase!* The Unix operating system is ‘‘case sensitive.’’ If you type your username in mixed case (`Rarmour` rather than `rarmour`, for example) the computer will not recognize it.

Your Password

Once you have typed in your username you will be prompted to type in your password. Type carefully! It won’t be displayed on the screen.

When you first login, you should change your password with the `yppasswd` command. Remember again-these are lower case commands and Unix insists that you type them that way.

Your password should be longer than six characters. It’s a good idea to make it mixed case or to stick some numbers or symbols in it, like ‘‘,’’ or ‘‘^’’. One of few password restrictions is that the password cannot be all-numeric (like 5534553). Because of a bug on the Sun computers, do not put a ‘‘:’’ in your password.

In the interests of self-preservation, *don’t* set your password to your username, to ‘‘password’’ or to any information which people are likely to know about you (your real name, your nickname, your pet dog’s name).

If you mistype your username or password you will get a suspicious message from the computer and see the `login:` prompt again.

The motd

If you type your username and password correctly, the computer will begin running the login program. It starts by displaying a special ‘‘message of the day’’---contained in the `/etc/motd` file. This file will usually contain information about the computer you are logging onto, maybe a basic message about getting help, and any important system messages from the system manager.

Initialization Files

When you log in the Unix login program finally starts up a command ‘‘shell.’’ Users do not deal with the operating system directly. Instead they interact with a shell, which is initialized with several pieces of information (such as your username, login directory and ‘‘path’’). By default all users use the C shell (the program `/bin/csh`) and interact with it.

There are a couple of files read by this shell when your login session starts up. These are the `.cshrc` file and the `.login` file. These files are created when your account is created. As you learn more about how Unix and the C shell work, you may want to customize these files.

If your files get corrupted for some reason, copies of the system defaults are available in /usr/local/skel/.

Using the System

Finally you are logged in! You will see a prompt like one of the following three:

```
pooh>

{coil:1}

%
```

just waiting for you to type something. Throughout the Unix Tutorial section we will use `%` to indicate the computer’s ‘‘ready’’ prompt.

ls

Okay, let’s try a simple command. Type `ls` and press . `ls` is the program to list files in a directory. Right now you may or may not see any files-not seeing any files doesn’t mean you don’t have any! Just plain `ls` won’t list hidden files (files whose names start with ‘‘.’’, like `.login`). Now try typing:

```
% ls -a
```

Don’t actually type the `%` symbol! Remember, that’s the computer’s prompt which indicates it is ready to accept input. The spacing should be exactly as shown. `ls` followed by a space, followed by a `-a.` The `-a` is a ‘‘flag’’ which tells the `ls` program to list all files.

For more about command flags see below.

cd

Just for fun, let’s look at the contents of another directory, one with lots of files. Directory names in Unix are straightforward. They are all arranged in a tree structure from the root directory ‘‘/’’.

For now, use `cd` to change your directory to the `/bin` directory. Type:

```
    % cd /bin
```

and press <CR>. Now type `ls` again. You should see a long list of files-in fact, if you look carefully you will see files with the names of the commands we’ve been typing (like `ls` and `cd`). Note that the `/bin` in the command we typed above was *not* a flag to `cd`. It was a ‘‘parameter.’’ Flags tell commands how to act, parameters tell them what to act on.

Now return to your login directory with:

```
    % cd
```

Entering `cd` with no parameter returns you to your home directory. You can check to make sure that it worked by entering:

```
    % pwd
```

which prints your current (or ‘‘working’’) directory. The computer should return a line of words separated by ‘‘/’’ symbols which should look something like:

```
    /home/username
```

Whatever it returns, the list should end in your username.

Using the On-line Man Pages

Most Unix commands have very short and sometimes cryptic names like `ls`. This can make remembering them difficult. Fortunately there are on-line manual pages which allow you to display information on a specific program (to list all the flags of `ls`, for example) or list all the information available on a certain topic.

man

To investigate other flags to the `ls` command (such as which flags will display file size and ownership) you would type `man ls`.

man -k

The second way of using the on-line manual pages is with `man -k`. In this case you use a word you expect to be in a one-line description of the command you wish to find. To find a program which ‘‘lists directory contents’’ you might type `man -k dir`. Partial words can be used and this is one of the few places in Unix where upper and lower case are allowed to match each other.

Using man and more

Try it now. Use `man ls` to find out how to make the `ls` program print the sizes of your files as well as their names. After typing `man ls` and pressing , note how `man` displays a screenful of text and then waits with a prompt `--More--` at the bottom of the screen.

What `man` is doing is sending everything it wants to display to the screen through a program known as a ‘‘pager’’ The pager program is called `more`. When you see `--More--` (in inverse video) at the bottom of the screen, just press the space-bar to see the next screenful. Press <CR> to scroll a line at a time.

Have you found the flag yet? The `-s` flag should display the size in kilobytes. You don’t need to continue paging once you have found the information you need. Press `q` and `more` will exit.

Listing File Sizes

Now type `ls -as`. You can stack flags together like this-this tells `ls` to list all files, even hidden files, and list their sizes in kilobytes.

Logging Off

When you are finished you should be sure to logout! You need to be careful that you’ve typed `logout` correctly. The Unix operating system is not forgiving of mis-typed commands. Mis-typing `logout` as ‘‘logotu’’, pressing return and then leaving without glancing at the screen can leave your files at anyone’s mercy.

Directory and File Structure

When you list files in Unix, it is very hard to tell what kind of files they are. The default behavior of the `ls` program is to list the names of all the files in the current directory without giving any additional information about whether they are text files, executable files or directories! This is because the ‘‘meaning’’ of the contents of each file is imposed on it by how you use the file. To the operating system a file is just a collection of bytes.

There is a program `file` which will tell you information about a file (such as whether it contains binary data) and make a good guess about what created the file and what kind of file it is.

File Names

Unlike other operating systems, filenames are not broken into a name part and a type part. Names can be many characters long and can contain most characters. Some characters such as * and ! have special meaning to the shell. They should not be used in filenames. If you ever do need to use such a symbol from the shell, they must be specified sneakily, by ‘‘escaping’’ them with a backslash, for example \!.

Directories

Directories in Unix start at the root directory ‘‘/’’. Files are ‘‘fully specified’’ when you list each directory branch needed to get to them.

```
/usr/local/lib/news

/home/pamela/src/file.c
```

The ‘‘File System’’ Tree Structure

Usually disks are ‘‘partitioned’’ into smaller sized sections called partitions If one partition of the disk fills up the other partitions won’t be affected.

Only certain large directory points are partitions and the choice of these points can vary among system managers. Partitions are like the larger branches of a tree. Partitions will contain many smaller branches (directories) and leaves (files).

The df Program

To examine what disks and partitions exist and are mounted, you can type the `df` command at the `%` prompt. This should display partitions which have names like `/dev/sd3g`---3 for disk 3, g for partition g. It will also display the space used and available in kilobytes and the ‘‘mount point’’ or directory of the partition.

Disk Space Maintenance

It’s important to keep track of how much disk space you are using. The command `du` displays the disk usage of the current directory and all of its subdirectories. It displays the usage, in kilobytes, for each directory-including any subdirectories it contains-and ends by displaying the total.

`% du`
: display disk usage of current directory

`% du -s`
: display only total disk usage

`% du -s -k`
: some versions of Unix need -k to report kilobytes

Scratch Space

Users have home directories for storing permanent files. At various busy times of the year there may be shortages of disk space on the Unix Cluster. You should use the `du` command to stay aware of how much space you are using and not exceed the system limits.

Your Login Directory

A login directory can always be specified with *~username* (~ is commonly called ‘‘twiddle,’’ derived from proper term ‘‘tilde.’’) If you needed to list files in someone else’s login directory, you could do so by issuing the command:

`% ls` *~username*

substituting in their username. You can do the same with your own directory if you've `cd`'d elsewhere. Please note-many people would consider looking at their files an invasion of their privacy; even if the files are not protected! Just as some people leave their doors unlocked but do not expect random bypassers to walk in, other people leave their files unprotected.

Subdirectories

If you have many files or multiple things to work on, you probably want to create subdirectories in your login directory. This allows you to place files which belong together in one distinct place.

Creating Subdirectories

The program to make a subdirectory is `mkdir.` If you are in your login directory and wish to create a directory, type the command:

```
    % mkdir directory-name
```

Once this directory has been created you can copy or move files to it (with the `cp` or `mv` programs) or you can `cd` to the directory and start creating files there.

Copy a file from the current directory into the new subdirectory by typing:

`cp` *filename directory-name/new-filename*
    copy file, give it a new name

`cp` *filename directory-name*
    copy file, filename will be the same as original

Or `cd` into the new directory and move the file from elsewhere:

```
    % cd directory-name
    % cp ../filename .
```

copies the file from the directory above giving it the same filename: ‘‘.’’ means ‘‘the current directory’’

Specifying Files

There are two ways you can specify files. Fully, in which case the name of the file includes all of the root directories and starts with ‘‘/’’, or relatively, in which case the filename starts with the name of a subdirectory or consists solely of its own name.

When Charlotte Lennox (username `lennox`) created her directory `arabella`, all of the following sets of commands could be used to display the same file:

```
    % more lennox/arabella/chapter1
or
    % cd lennox
    % more arabella/chapter1
or
```

```
% cd lennox/arabella
% more chapter1
```

The full file specification, beginning with a ‘‘/’’ is very system dependent. On oceanography machines, all user directories are in the `/usra` partition.

```
/usra/lennox/arabella/chapter1
```

Protecting Files and Directories

When created, all files have an owner and group associated with them. The owner is the same as the username of the person who created the files and the group is the name of the creator’s default login group, such as `users, system` etc. Most users do not belong to a shared group on our systems. If the creator of the file belongs to more than one group (you can display the groups to which you belong with the `groups` command) then the creator can change the group of the file between these groups. Otherwise, only the root account can change the group of a file.

Only the root account can change the ownership of a file.

Displaying owner, group and protection

The command `ls -lg` *filename* will list the long directory list entry (which includes owner and protection bits) and the group of a file.

The display looks something like:

```
protection  owner       group      filename
-rw-r-----  hamilton    ug         munster_village
```

The Protection Bits

The first position (which is not set) specifies what type of file this is. If it were set, it would probably be a `d` (for directory) or `l` (for link). The next nine positions are divided into three sets of binary numbers and determine protection to three different sets of people.

```
 u      g      o
rw-    r--    ---
 6      4      0
```

The file has ‘‘mode’’ 640. The first bits, set to ‘‘r + w’’ (4+2) in our example, specify the protection for the user who owns the files (u). The user who owns the file can read or write (which includes delete) the file.

The next trio of bits, set to 4, or ‘‘r,’’ in our example, specify access to the file for other users in the same group as the group of the file. In this case the group is ug-all members of the ug group can read the file (print it out, copy it, or display it using `more`).

Finally, all other users are given no access to the file.

The one form of access which no one is given, even the owner, is ‘‘x’’ (for execute). This is because the file is not a program to be executed-it is probably a text file which would have no meaning to the computer. The x would appear in the 3rd position and have a value of 1.

Changing the Group and the Protection Bits

The group of a file can be changed with the `chgrp` command. Again, you can only change the group of a file to a group to which you belong. You would type as follows:

```
    % chgrp groupname filename
```

You can change the protection mode of a file with the `chmod` command. This can be done relatively or absolutely. The file in the example above had the mode 640. If you wanted to make the file readable to all other users, you could type:

```
    % chmod 644  filename
or
    % chmod +4  filename    (since the current mode of the file was 640)
```

For more information see the man page for `chmod`.

Default Protections: Setting the umask

All files get assigned an initial protection. To set the default initial protection you must set the value of the variable `umask. umask` must be defined once per login (usually in the `.cshrc` file). Common umask values include 022, giving read and directory search but not write permission to the group and others and 077 giving no access to group or other users for all new files you create.

The Unix Shell Syntax

As mentioned earlier, user commands are parsed by the shell they run. There are many shells other than the the C shell which allow different types of shortcuts. We will only discuss the C shell here, but some alternate shells include the Bourne shell (`/bin/sh`), the Bourne-Again Shell (`bash`), `zsh` and `tcsh` (a C shell variant).

The Path

One of the most important elements of the shell is the path. Whenever you type something at the `%` prompt, the C shell first checks to see if this is an ‘‘alias’’ you have defined, and if not, searches all the directories in your path to determine the program to run.

The path is just a list of directories, searched in order. Your default `.cshrc` will have a path defined for you. If you want other directories (such as a directory of your own programs) to be searched for commands, add them to your path by editing your `.cshrc` file. This list of directories is stored in the PATH environment variable. We will discuss how to manipulate enviroment variables later.

Flags and Parameters

Most commands expect or allow parameters (usually files or directories for the command to operate on) and many provide option flags. A ‘‘flag’’ as we saw before, is a character or string with a `-` before it-like the `-s` we used with the `ls` command.

Some commands, such as `cp` and `mv` require file parameters. Not surprisingly, `cp` and `mv` (the copy and move commands) each require two! One for the original file and one for the new file or location.

It would seem logical that if `ls` by itself just lists the current directory then `cp` *filename* should copy a file to the current directory. This is logical-but wrong! Instead you must enter `cp` *filename* . where the ‘‘.’’ tells `cp` to place the file in the current directory. *filename* in this case would be a long filename with a complete directory specification.

Not surprisingly `ls .` and `ls` are almost the same.

Creating Files

The cat Program

`cat` is one of most versatile commands. The simplest use of `cat`:

```
% cat .cshrc
```

displays your `.cshrc` file to the screen. Unix allows you to redirect output which would otherwise go to the screen by using a > and a filename. You could copy your `.cshrc`, for example, by typing:

```
% cat .cshrc > temp
```

This would have the same effect as:

```
% cp .cshrc temp
```

More usefully `cat` will append multiple files together.

```
% cat .cshrc .login > temp
```

will place copies of your `.cshrc` and `.login` into the same file. Warning! Be careful not to cat a file onto an existing file! The command:

```
% cat .cshrc > .cshrc
```

will *destroy* the file `.cshrc` if it succeeds.

If you fail to give `cat` a filename to operate on, cat expects you to type in a file from the keyboard. You must end this with a <Ctrl>-D on a line by itself. <Ctrl>-D is the end-of-file character.

By combining these two-leaving off the name of a file to input to `cat` and telling `cat` to direct its output to a file with > *filename*, you can create files.

For example:

```
    % cat > temp

    ;klajs;dfkjaskj
    alskdj;kjdfskjdf
    <Ctrl>-D
    %
```

This will create a new file `temp`, containing the lines of garbage shown above. Note that this creates a new file-if you want to add things on to the end of an existing file you must use `cat` slightly differently. Instead of > you'd use >> which tells the shell to append any output to an already existing file. If you wanted to add a line onto your `.cshrc`, you could type

```
    % cat >> .cshrc
    echo "blah blah blah"
    <Ctrl>-D
    %
```

This would append the line `echo "blah blah blah"` onto your `.cshrc`. Using > here would be a bad idea-it might obliterate your original `.cshrc` file.

Text Editors

`cat` is fine for files which are small and never need to have real changes made to them, but a full fledged editor is necessary for typing in papers, programs and mail messages. Among the editors available `pico`, `vi` and `emacs.`

Be careful! Not all Unix editors keep backup copies of files when you edit them.

pico

`pico` is a simple, friendly editor--the same editor as used in pine. Type `pico` *filename* to start it and type `man pico` for more information about how to use it.

vi

`vi` is an editor which has a command mode and a typing mode. When you first startup `vi` (with the command `vi` *filename*) it expects you to enter commands. If you actually want to enter text into your file, you must type the insert command `i.` When you need to switch back to command mode, hit the escape key, usually in the upper left corner of your keyboard.

To move around you must be in command mode. You can use the arrow keys or use `j, k, h, l` to move down, up, left and right.

For more information type `man vi`. There are two reference sheets containing lists of the many `vi` commands available from C&C (located at Brooklyn and Pacific).

Emacs

Emacs is a large editing system. Copies of the manual are for sale at the CCO Front Desk and copies of the two-page reference sheet are available in the reference sheet rack across from the Front Office.

To use `emacs`, type:

```
% setup emacs
% emacs
```

Files as Output and Log Files

Ordinarily there are two types of output from commands: output to standard output (stdout) and to standard error (stderr). The > and >> examples above directed only standard output from programs into files. To send both the standard output and error to a file when using the C shell, you should type `>&` :

```
% command >& filename
```

Logging Your Actions to a File

Sometimes you may wish to log the output of a login session to a file so that you can show it to somebody or print it out. You can do this with the `script` command. When you wish to end the session logging, type `exit`.

When you start up you should see a message saying `script started, file is typescript` and when you finish the script, you should see the message `script done`. You may want to edit the typescript file-visible ^M's get placed at the end of each line because linebreaks require two control sequences for a terminal screen but only one in a file.

Comparing Files

The basic commands for comparing files are:

`cmp`
: states whether or not the files are the same

`diff`
: lists line-by-line differences

`comm`
: three column output displays lines in file 1 only, file 2 only, and both files

See the man pages on these for more information.

Searching Through Files

The `grep` program can be used to search a file for lines containing a certain string:

```
% grep  string filename
% grep -i  string filename      (case insensitive match)
```

or not containing a certain string:

```
% grep -v string filename
```

See the man page for `grep`---it has many useful options.

`more` and the `vi` editor can also find strings in files. The command is the same in both-type a */string* when at the `--More--` prompt or in `vi` command mode. This will scroll through the file so that the line with ‘‘string’’ in it is placed at the top of the screen in `more` or move the cursor to the string desired in `vi.` Although `vi` is a text editor there is a version of `vi, view,` which lets you read through files but does not allow you to change them.

The System and Dealing with Multiple Users

Most Unix commands which return information about how much CPU-time you’ve used and how long you’ve been logged in use the following meanings for the words ‘‘job’’ and ‘‘process.’’

When you log in, you start an interactive ‘‘job’’ which lasts until you end it with the `logout` command. Using a shell like C shell which has ‘‘job-control’’ you can actually start jobs in addition to your login job. But for the purposes of the most information returning programs, `job` (as in the ‘‘JCPU’’ column) refers to your login session.

Processes, on the other hand, are much shorter-lived. Almost every time you type a command a new process is started. These processes stay ‘‘attached’’ to your terminal displaying output to the screen and, in some cases (interactive programs like text editors and mailers) accepting input from your keyboard.

Some processes last a very long time-for example the `/bin/csh` (C shell) process, which gets started when you login, lasts until you logout.

Information about Your Processes

You can get information about your processes by typing the `ps` command.

```
  PID TT STAT  TIME COMMAND
 9980 s9 S     0:06 -csh (csh)
12380 s9 R     0:01 ps
```

The processes executing above are the C shell process and the `ps` command. Note that both commands are attached to the same terminal (TT), have different process identification numbers (PID), and have different amounts of CPU-time (TIME), accumulated.

Information about Other People’s Processes

who

The simplest and quickest information you can get about other people is a list of which users are logged in and at which ‘‘terminals’’ (terminal here is either a terminal device line or telnet or rlogin session).

The command to do this is `who` and it responds quickest of all the commands discussed here because it simply examines a file which gets updated everytime someone logs in or out.

Be careful though! This file, `utmp,` can get out of date if someone's processes die unexpectedly on the system. Any program which uses `utmp` to report information may list users who are not really logged in!

w

The `w` command is slower than the `who` command because it returns more information such as details about what programs people are running. It also returns a line containing the number of users and the system load average. The load average is the average number of processes ready to be run by the CPU and is a rough way of estimating how busy a system is.

`w` also uses the `utmp` file mentioned above. It takes longer than `who` because it then looks around and collects more information about the users it finds in the `utmp` file.

ps

The `ps` command used earlier to list your own processes can be used to list other users' processes as well. `who` and `w` list logins-but not individual processes on the system. They don't list any of the running operating system processes which start when the computer is booted and which don't have logins.

Since `ps` doesn't use `utmp` it is the program to use when you really want to find out what processes you might have accidentally left on the system or if another user is running any processes. Note that although `ps` might report processes for a user, it might be because that user has left a ‘‘background job’’ executing. In this case you should see a ‘‘?’’ in the TT field and the user won't really be logged in.

To get this fuller listing, give the flags `-aux` to `ps`. For more information on the uses of `ps,` type `man ps`.

finger

The `finger` program returns information about other users on the system who may or may not be logged in. `finger` by itself returns yet another variation of the list of currently logged in users. `finger` followed by a username or an e-mail -style address will return information about one or more users, the last time they logged into the system where you are fingering them, their full name, whether or not they have unread mail and, finally, the contents of two files they may have created: `.plan` and `.project`

For more information about using `finger` or ways to provide information about yourself to others, type `man finger`.

Sending Messages and Files to Other Users

Electronic mail programs run on almost all the computers at Caltech and usually have two parts: a user interface which lets users read and send messages and a system mailer which talks to mailers on other computers. This mailer receives outgoing messages from the user interface programs and delivers incoming messages to the user mailbox (which the interface program reads).

/usr/ucb/mail

There are many user interfaces available on the Unix computers, all of which provide similar functionality. The program supplied with most Unix computers is `/usr/ucb/mail` (or `Mail`). To read messages type `Mail`, to send messages type:

```
% Mail address
```

Mail has been changed to `mailx`.

You should next see a `Subject:` prompt. If you don't see a prompt, don't worry, just type in your one line subject anyway and press return. You may start typing your message (but you will be unable to correct errors on lines after you have pressed <CR> to move to the next line) or you may may specify a file to include with `r` *filename*.

You may invoke a text editor like `vi` by typing `v`. If you wish regularly to use an editor other than `vi` you should see the information later in this section about enviroment variables.

There are many other commands you may enter at this point-see the `Mail` man page for all of them. When you are finished typing in your message (if you have used `v` to run a text editor, you should exit from it) press <Ctrl>-D on a line by itself. Most likely you will now see a `CC:` prompt. If you wish to send copies of your message to someone besides the recipient you would enter the address or addresses (separated by ‘‘,’’) and press return. Otherwise press return without entering an address.

PINE

PINE is a full-screen interactive mailer, developed at UW, that is very straightforward to use. To use it type `pine`. More information is available from the UW C&C web server.

Write

The `write` program can be used to send messages to other users logged onto the system. It's not a great way of having a conversation, but it's simple to use. Enter:

```
% write username
```

and you can start writing lines to the terminal of the person you want to send messages to. The person must be logged in, and, if they are logged in more than once, you must specify the terminal to `write` to-for example `write melville ttyh1`.

Talk

`talk` is a program which allows two users to hold a conversation. Unlike `write`, it can be used between different computers; and, unlike `write`, it divides the screen so that the things you type appear in the top half and the things written to you appear in the bottom half.

To `talk` to users on the same computer:

```
% talk username
```

To `talk` to users on another computer use the address format of *username@nodename*:

```
% talk brunton@jarthur.claremont.edu
```

Addressing Remote Nodes

`talk` can only be used to other Internet nodes-computers which usually have ending names such as .edu, .com, .org, .gov, or .mil. Not all computers with these names are attached directly to the Internet---`finger` and `talk` won't work with computers which are only attached by mail gateways.

Shortcuts

If you use certain command flags regularly (`-lga` for `ls`) you can `alias` them to shorter commands. You can use wildcard symbols to refer to files with very long names. You can easily repeat commands you have already executed or modify them slightly and re-execute them.

Aliases

As mentioned above, you can `alias` longer commands to shorter strings. For example, `ls -F` will list all the files in the current directory followed by a trailing symbol which indicates if they are executable commands (a *) or directories (a /). If you wanted this to be the default behavior of ls you could add the following command to your `.cshrc`:

```
% alias ls ls -F
```

To list the aliases which are set for your current process, type:

```
% alias
```

without any parameters.

Wildcards

Wildcards are special symbols which allow you to specify matches to letters or letter sequences as part of a filename.

Some examples:

*
: The basic wildcard character. Beware `rm *`!!
`ls *.dat`

lists all files ending in `.dat`

`ls r*`

lists all files starting with `r`

?

a one character wildcard.

`ls ?.dat`

lists `5.dat`, `u.dat`, but not `70.dat`

[]

limits a character to match one of the characters between the brakets

`ls *.[ch]`

lists all `.h` and `.c` files

`more [Rr][Ee][Aa][Dd][Mm][Ee]`

`more`s the files `README`, `readme`,`ReadMe`, and `Readme`, among others

Directory Specifications

You've already met the shortcut. The two other important directory symbols are ''.'' for the current directory and ''..'' for the previous (parent) directory.

```
% cd ..
```

moves you out of a subdirectory into its parent directory.

Environment Variables

Environment variables are pieces of information used by the shell and by other programs. One very important one is the PATH variable mentioned earlier. Other important variables you can set include:

- EDITOR
- TERM
- MAIL

To see what environment variables are set and what they are set to, type the command `printenv`. To set a variable, use the `setenv` command as in the example below.

```
% setenv TERM vt100
% setenv EDITOR emacs
```

Many programs mention environment variables you may want to set for them in their man pages. Look at the `csh` man page for some of the standard ones.

History

Most shells allow ''command line editing'' of some form or another-editing one of the previous few lines you've typed in and executing the changed line. You can set a history ''environment variable'' to determine how many previous command lines you will have access to with `set history=40`

Repeating and Modifying the Previous Command

The simplest form of command line editing is to repeat the last command entered or repeat the last command entered with more text appended.

If the last command you typed was:

```
% ls agreen
```

Then you can repeat this command by typing:

```
% !!
```

This will return a list of files. If you saw a directory `leavenworth` in the list returned and you wanted to list the files it contained, you could do so by typing:

```
% !!/leavenworth
```

If you mistype `leavenworth` as `leaveworth` you can correct it with the following command:

```
% ^leave^leaven
```

This substitutes leaven for leave in the most recently executed command. Beware! This substitutes for the *first* occurrence of leave only!

Repeating Commands From Further Back in History

You can type `history` at any time to get a list of all the commands remembered. This list is numbered and you can type ! *number* to repeat the command associated with number. Alternately you can type ! and a couple of letters of the command to repeat the last line starting with the characters you specify. `!ls` to repeat `ls -lg agreen`, for example.

The .login and .cshrc Files

The `.cshrc` file is run whenever a C shell process is started. Then, if this is a login process, the `.login` file is executed. If you are using a NeXT console with a program such as Terminal, you can usually choose whether you want each new window to execute the `.login` file by making a change to your Preferences in the Terminal program's Preferences menu. By default the `.login` will get executed.

If you are using a Sun console and you have the default setup, any xterm windows which you start up will not execute the `.login`.

Job Control

It is very easy to do many things at once with the Unix operating system. Since programs and commands execute as independent processes you can run them in the ‘‘background’’ and continue on in the foreground with more important tasks or tasks which require keyboard entry.

For example, you could set a program running in the background while you edit a file in the foreground.

The fg and bg Commands

When you type <Ctrl>-Z whatever you were doing will pause. If you want the job to go away without finishing, then you should kill it with the command `kill %`. If you don't want it paused but want it to continue in the foreground-that is, if you want it to be the primary process to which all the characters you type get delivered-type `fg`. If you want it to continue processing in the background while you work on something else, type `bg`.

You should not use `bg` on things which accept input such as text editors or on things which display copious output like `more` or `ps`.

What to Do When You've Suspended Multiple Jobs

If you've got several processes stopped-perhaps you are editing two files or you have multiple `telnet` or `rlogin` sessions to remote computers-you'll need some way of telling `fg` which job you want brought to the foreground.

By default `fg` will return you to the process you most recently suspended. If you wanted to switch processes you would have to identify it by its job number. This number can be displayed with the `jobs` command. For example:

```
    % jobs
    [1]          Stopped     vi .login
    [2]    +     Stopped     rn
    [3]          Running     cc -O -g test.c
    %
```

The most recently suspended job is marked with a + symbol. If you wanted to return to job one instead, you would type:

```
    % fg %1
```

You can type `%1` as a shortcut.

Starting Jobs in the Background

Some jobs should start in the background and stay there-long running compilations or programs, for example. In this case you can direct them to the background when you start them rather than after they have already begun. To start a job in the background rather than the foreground, append an `&` symbol to the end of your command.

You should always run background processes at a lower priority by using the `nice` command. Non-interactive jobs are usually very good at getting all the resources they need. Running them at a lower priority doesn't hurt them much-but it *really* helps the interactive users-people running programs that display to terminal screens or that require input from the keyboard.

If you need to run CPU-intensive background jobs, learn about how to control the priority of your jobs by reading the manual pages (`man nice` and `man renice`).

Suspend, z and <Ctrl>-Z

Some programs provide you with special ways of suspending them. If you started another shell by using the `csh` command, you would have to use the `suspend` command to suspend it.

If you wish to suspend a `telnet` or `rlogin` session you must first get past the current login to get the attention of the `telnet` or `rlogin` program.

Use (immediately after pressing a return) to get `rlogin`'s attention. <Ctrl>-Z will suspend an `rlogin` session.

Use <Ctrl>-] to get `telnet`'s attention <Ctrl>-]z will suspend a telnet session. Watch out, though, if you are connected from a PC with through Kermit! <Ctrl>-] is Kermit's default escape sequence. You'll need to type <Ctrl>-] `z` or define Kermit's escape sequence to something else such as <Ctrl>-K.

Some Common and Useful Unix Commands For Files

cp

The `cp` command allows you to create a new file from an existing file. The command line format is:

> `% cp` *input-file-spec output-file-spec*

where *input-file-spec* and *output-file-spec* are valid Unix file specifications. The file specifications indicate the file(s) to copy from and the file or directory to copy to (output). Any part of the filename may be replaced by a wildcard symbol (*) and you may specify either a filename or a directory for the *output-file-spec*. If you do not specify a directory, you should be careful that any wildcard used in the *input-file-spec* does not cause more than one file to get copied.

```
% cp new.c old.c
% cp new.* OLD (where OLD is a directory)
```

ls

command allows the user to get a list of files in the current default directory. The command line format is:

> `% ls` *file-spec-list*

where *file-spec-list* is an optional parameter of zero or more Unix file specifications (separated by spaces). The file specification supplied (if any) indicates which directory is to be listed and the files within the directory to list.

lpr

The `lpr` command tells the system that one or more files are to be printed on the default printer. If the printer is busy with another user's file, an entry will be made in the printer queue and the file will be printed after other `lpr` requests have been satisfied. The command line format is:

BLOCKQUOTE> `% lpr` *file-spec-list*

where *file-spec-list* is one or more Unix files to be printed on the default printer. Any part of the filenames may be replaced by a wild card.

Here is more information about where the printers actually are and what kind of printers are available.

man

The `man` command is a tool that gives the user brief descriptions of Unix commands along with a list of all of the command flags that the command can use. To use `man,` try one of the following formats:

```
    % man  command
    % man -k  topic
```

more

The `more` command will print the contents of one or more files on the user's terminal. The command line format is:

```
    % more  file-spec-list
```

`more` displays a page at a time, waiting for you to press the space-bar at the end of each screen. At any time you may type `q` to quit or `h` to get a list of other commands that `more` understands.

mv

The `mv` command is used to move files to different names or directories. The command line syntax is:

```
    % mv input-file-spec output-file-spec
```

where *input-file-spec* is the file or files to be renamed or moved. As with `cp`, if you specify multiple input files, the output file should be a directory. Otherwise *output-file-spec* may specify the new name of the file. Any or all of the filename may be replaced by a wild card to abbreviate it or to allow more than one file to be moved. For example:

```
    % mv data.dat ./research/datadat.old
```

will change the name of the file `data.dat` to `datadat.old` and place it in the subdirectory `research`.

Be very careful when copying or moving multiple files.

rm

The `rm` command allows you to delete one or more files from a disk. The command line format is:

> `% rm` *file-spec-list*

where *file-spec-list* is one or more Unix file specifications, separated by spaces, listing which files are to be deleted. Beware of `rm *`! For example:

```
% rm *.dat able.txt
```

will delete the file `able.txt` and all files in your current working directory which end in `.dat`. Getting rid of unwanted subdirectories is a little more difficult. You can delete an empty directory with the command `rmdir` *directory-name* but you cannot use `rmdir` to delete a directory that still has files in it.

To delete a directory with files in it, use `rm` with the `-r` flag (for recursive).