

## **A Crash Course in UNIX**

### **A Gentle Introduction for Those Who Have Never Used a Command Line**

Logging in using SSH.

See Jesse's presentation:

<http://www.stat.uga.edu/computing/xserver.html>

X Windows commands (via Xming): `xterm`, `xclock`, `xcalc`,  
`xemacs`, `matlab`, ...

See Jesse's presentation as well as a little of mine in the pages below.

`man command`; `man -k keyword`

This command gives you access to the manual pages. If there is a command that you need to know how to use, the man pages will tell you more than you want to know.

Directories in the UNIX environment are the same as folders on a PC.

`pwd`

Often, the command line will contain information about what directory you're currently in. If you need to know what directory you're in and there is no obvious clue, then use this command, which is short for `print working directory`.

`ls; ls -al`

This command, short for `list`, lists all the non-hidden files and directories in the current directory.

Hidden files are preceded by a `."`. For example, the file `.login` contains commands that are executed every time you log in. To see these files, you will need to use the option `-a` with the `ls` command. To see file details, such as size and access permissions, use the option `-l`. To do both at once, just use them as `-al`, rather than listing the two options separately, as `-a -l`.

`mkdir dirname`

To make a new directory, use this command.

`rmdir dirname`

To remove an empty directory, use this command. Yes, you must delete or move all files in that directory before the system will allow you to delete it.

`cd dirname; cd ..`

Use this command to change directory. If you are several layers of directories into your file system, and you want to go one directory up, then `..` is shorthand for the next directory up (whatever it may be).

**ASIDE:** If the command history is available, then you can use the up and down arrows to repeat commands. This is very handy when you have to execute a lengthy command several times!

On a PC, you use the mouse to move, copy, and rename files. You will need commands to do this in a UNIX environment.

```
mv oldfile newfile
```

This command may be used to rename a file (from *oldfile* to *newfile*), or to move a file to another directory (in which case, the *newfile* name will have to include the directory path).

```
cp file1 file2
```

This command may be used to copy a file into another file (which may be in another directory).

```
rm filename; rm -i filename
```

Use this command to remove (delete) a file. NOTE: There is no Recycle Bin -- once you have removed a file, it is GONE! The `-i` option asks you if you *really* want to delete the file.

ASIDE: The \* is a wildcard that may substitute for all or part of a file name. If you use the command `rm *`, you will remove everything. If you use `rm w*`, you will remove all files beginning with a *w*.

```
more txtfile; less txtfile
```

This commands lists a text (or, ASCII) file one page at the time, where the page size is defined by your screen size. There aren't too many text files these days, except of course for your program files, but this command is useful in combination with other commands that produce a LOT of lines...

```
command1 | command2
```

This useful little trick, the simple vertical line, is called "pipe". It pipes (or, filters) one command through another. For example, we've seen that `ls -al` will give a detailed list of all of your files, including the hidden ones. If you have pages of them, you can't find the file or directory name you're looking for! So, you pipe the command through `more`: `ls -al | more`, to see the contents of your directory one page at a time.

To print, you can either transfer the file to your PC and print in the way you're accustomed to, or you can do so from the command line.

```
lp filename; lp filename -d server:printername
```

This command harkens back to the olden days when we had line printers -- it simply sends the file named to be printed.

As with PCs, there is a default printer associated with the UNIX system you're on. If you want to print elsewhere, you need to know the name of that printer. For example, to print to the printer in the copy room, you would execute the command `lp filename -d scheme:lw`. From `rcluster`, you should be able to print to the printer in the graduate student offices using `lp filename -d scheme:gradsprn`.

Note: It is easier to use the PC printer window, to cancel a printer job. If you need to use the command line, you will need the command `lpq` to identify the job number, and the command `cancel` to actually cancel that job. Use the `man` command to find out more about these commands.

Sometimes you need to clean up your files because you're running out of disk space. One way to clean up is by compressing your files.

```
du; du -sk
```

This command tells you about how much disk space your current directory uses. If you name a directory, it will tell you about that specific directory. The `-s` option gives you a summary of the space used by the contents of the directory. The `-k` option gives the space usage in units of kilobytes.

```
gzip filename; gzip -9 filename
```

This command compresses the given file. The option `-9` compresses that file as small as it can possibly get. If you use the wildcard `*` in place of the file name, then all files in that directory will be compressed. The compressed file will be renamed *filename.gz*.

```
gunzip filename
```

Guess what this one does??

tar

I have to go to the `man` pages every time I use this command. This one creates compressed archives of files and directories, and extracts files or directories from archives. The archives created have the extension `.tar`. Options will give a list of contents, add files, extract files, etc.

Other neat little commands are:

who

Tells you who is logged on to the system.

`finger username`

Gives information about the named user. The user name can be the login name, first name, or last name.

logout

Guess.

```
alias shortcut command; alias shortcut 'commandstring'
```

This command allows you to make up your own commands as substitutes for others. For example, `alias bye logout` will allow you to use `bye` in place of `logout`. You can even make DOS commands: `alias cls clear; or alias dir 'ls -al | more'`. Warning: It is a little different on `rcluster`! E.g., `alias shortcut=command`.

```
ps; ps -fu userid; ps -ef | more
```

This command shows the processes, including programs, that are currently running. The option `-f` gives a fully detailed listing; `-u` gives only the processes for the given login name; and `-e` requests every process running; .

### Control-Key Commands:

These are helpful, especially for frozen programs that you've run from the command line! See [http://web.cecs.pdx.edu/~rootd/catdoc/guide/TheGuide\\_38.html#SEC41](http://web.cecs.pdx.edu/~rootd/catdoc/guide/TheGuide_38.html#SEC41) for lots of handy control-key commands.

`grep pattern file`

This command searches the named file for the given pattern. You can also pipe output of another command through `grep` to search for instances of a word or phrase of interest.

Text Editors: `xemacs`, `emacs`, `pico`, `vi`

If you're going to write a program in a UNIX environment, you will need a text editor. The options I've given are listed in order of ease of use. Only `vi` is guaranteed to be available on every UNIX system! More on `vi` may be found by Googling "vi tutorial."

`source`, writing scripts

You can put a bunch of UNIX commands in a file to execute all at once. If you put them in the file named `cleanup`, then you run those commands with the command `source cleanup`. This file of UNIX commands to be executed at once like a program is called a *script*.

```
kill job#; kill -9 job#
```

If a command or program is hung up in the sense that you can't stop it from running with a control command or anything else, then use the `kill` command to stop it. You must use `ps` to identify the job number. You cannot kill a process that you do not own. If `kill` by itself does not work, then `kill -9` (translated, "kill it until it is dead!") will work.

```
command &
```

When you want to run a command in the background (that is, you want to use your command line to do other things while that command is executing), you put `&` at the end of it. This is extremely useful when you're running MatLab, or a very lengthy simulation! Note: You should be able to log out while a program is quietly executing in the background.

```
command > outfile
```

The character `>` will redirect input into or output from a command. We will see an example next.

```
cat file1 ... filen > bigfile
```

This file concatenates (stacks together, with the first on top and the last on bottom) the named files. Without the redirection, the output would simply scroll down your screen. With the redirection, the output is placed into another file. This is handy when you need pieces from several programs to form a new program.

```
head, tail
```

Sometimes, you only need to see a piece of a file rather than the whole thing. The command `head` shows you the first 10 lines of a file; the command `tail` shows you the last 10. You can change that number with options.

```
dos2unix winname > unixname
```

When you edit or create a text file in Windows, the result contains characters that you can't see, and that often your UNIX editor can't see. These control characters at the end of each line are removed by using this command.

`time command`

This returns the time it takes in order to execute the command. So, you can time your programs this way to compare computing speed. The information returned includes CPU time (the actual time that the computer spent executing your program) as well as the actual time taken to run the program (this may be longer, especially if there are others running jobs at the same time).

Batch commands for R and MatLab on `rcluster`.

These sequences of commands will run batch jobs on `rcluster`. Note that some of these commands, especially the `b*` commands, don't work on our UNIX machines!

To run an R program, use a text editor on `rcluster` to create a file containing these lines:

```
#!/bin/sh
time R CMD BATCH RprogramName
exit
```

and save this script into a file named, say, `sub.sh`. Submit this file to the batch queue with the following 2 commands:

```
chmod u+x sub.sh
```

```
bsub -q r1-10d -o sub.%J.out -e sub.%J.err ./sub.sh
```

In the script:

- The first line tells the computer to use a Bourne shell (rcluster uses a bash shell).
- The second submits your R program to the batch queue, and times it.
- The third command exits the Bourne shell.

Of the 2 commands:

- The first changes `sub.sh` into an executable file.
- The second runs the script:
  - In the queue named `r1-10d` (this one has a maximum runtime of 10 days; others, like `r1-24h`, has a runtime of 24 hours).
  - With output going to the file `sub.%J.out` and errors going to the file `sub.%J.err`.
  - Where the symbol `%J` will be replaced by the jobnumber for the program.

Handy batch job commands that we don't have on `plot`, but we do have on `rcluster`:

`bjobs`

This command shows you all of the jobs you have running, including the job ID number. You'll need that if you need to kill your program...

`bkill -9 job#`

This command kills (until it is dead! ;-)) the batch job with job ID number `job#`. Note that you can't kill a job that you don't own.

`bkill -u yourusername`

This command kills all of your batch jobs.

## Exercise

If you put these commands

```
#!/bin/sh
```

```
matlab -nodisplay < filename.m > out.txt
```

```
exit
```

into a file you name `sub.sh`, and then run the commands

```
chmod u+x sub.sh
```

```
bsum -q r1-10d -o sub.%J.out -e sub.%J.err ./sub.sh
```

you will run a MatLab program called `filename` in batch mode. Tell what each command is doing, including what the options are doing.