

The “twelve” most important Unix commands

Carl Mason
carlm@demog.berkeley.edu

rev 1.2 Fall 2011

Contents

1	Introduction	1
2	Terminal windows	2
3	The Filesystem	3
4	The command interpreter	5
4.1	Essential stuff	5
4.2	Efficient stuff	6
5	the 12 most important Unix commands	7
6	Special and “meta” characters	11

1 Introduction

Although Unix has a point and click graphic user interface, called X11, which works just like those other operating systems, Unix is at heart a command line operating system. So while it is possible in many cases to do what you want via pointing and clicking, using the command line and other text based tools will make you happier **much** more efficient... eventually.

To operate with the command line, you will need to know the 12 most important Unix commands described in Section 5. To enjoy it you will also need to know a few tricks that are also covered in this document.

You don't **need** to know much about Unix in order to start doing Science, but it would not hurt to learn more. In your copious free time, check out

some of the Unix primers on the web. Ask google something like “Unix beginner” to find more resources than you could possibly want.

Note that since the Mac OS is simply a Unix application, nearly everything in this document works the same way in a mac. On a mac, the terminal window application is under **Applications/Utilities**.

2 Terminal windows

In order to use the command line or *shell*, you must open a *terminal window* (also known as an *xterm* window). A terminal window can be launched from: [Application] → [System Tools] → [Terminal].

It should look something like Figure 1. Notice that the window features a menu bar, unfortunately the menu bar it is both useless and misleading. Make the menu bar disappear by pressing the [RIGHT BUTTON] and clearing the “show Menubar” check-box.

Now aside from the title bar at the top, the only words in the terminal window should be the *Unix prompt*. The purpose of the Unix prompt is to indicate that the *shell* is ready to accept commands. It also contains useful information. In Figure 1, the prompt is [carlm@twins ~]\$, indicating the user, carlm, the machine, twins and the current directory which is indicated by the ~. In this and other documents, the Unix prompt will look like this: @:>. In the real Unix prompt, the symbol ~ is a special character whose meaning is “home directory”. ~/Dissertation means a file or directory called “Dissertation” which is located within your home directory. In my case this would be /hdir/0/carlm/Dissertation. ~wachter/Brilliant/insight translates to a file (or possibly a directory) called insight in a directory¹ called Brilliant in Ken Wachter’s home directory, or /hdir/0/wachter/Brilliant/insight. More about home directories can be found in Section 2.

If you are in no particular hurry to finish your dissertation, you can modify a large number of colors and beeps and other important features of the terminal window. [RIGHT BUTTON] [Edit Current Profile] is the place to start wasting time.

If you have already wasted time on this sort of thing and are thus old enough to find the default font a bit small, a useful trick with terminal windows (and browser windows too) is the [CTRL] + [SHIFT] + [+] to increase and [CTR] + [-] to decrease the size of the typeface.

¹directories are also called “folders”

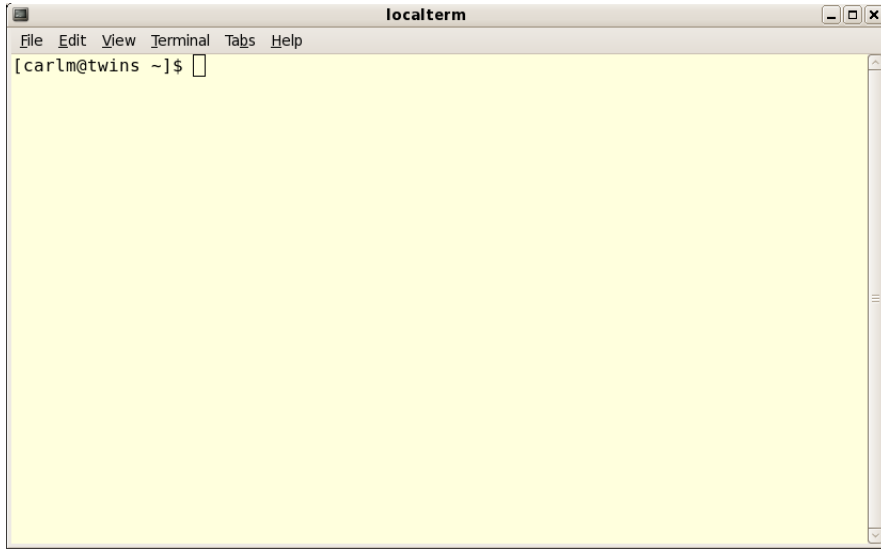


Figure 1: terminal window

3 The Filesystem

Whenever you login to a machine on the Demography network, your initial *present working directory* – the location within the filesystem in which applications will begin looking for the files that you specify – is your home directory. Every user has exactly one home directory.

In a multiuser system such as the Demography Lab, your home directory is one of a huge number of interconnected directories that form a single unified *filesystem*. The magic of the filesystem is that even though the various files and directories of which it is composed are physically present on various different machines all over the network, to us users, the whole thing appears to be one single thing and that thing looks and feels the same no matter which Demography Lab machine we happen to be using at the moment.

An upside down tree makes a pretty good metaphor for the filesystem. Such a “tree” is shown in Figure 2. At the top of the figure is a directory called “/” which is the “root” of the filesystem. Every file and directory in the filesystem can be uniquely specified by a *filepath* that begins with root. For example, the file that holds my correspondence with my mother is `/hdir/0/carlm/mail/mom`.

As you can see in Figure 2 home directories all live in a directory called

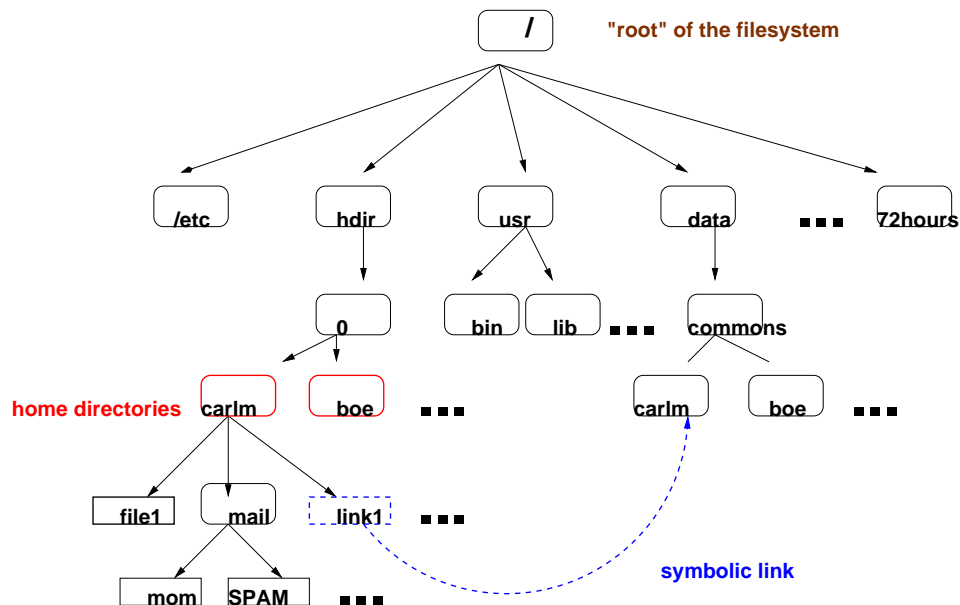


Figure 2: The Demography Lab filesystem

/hdir/0. Although it is just one of many directories within this giant upside down tree of a filesystem, your home directory is a special place that you will come to know and love and where you will do your very best work. It is the part of the filesystem that you own and the “place” where you will find yourself when you first login.

Because the entire filesystem looks the same to all users all the time, it is easy to share data with your colleagues. This is good thing because humanity benefits when scientists collaborate. But unfortunately scientists can occasionally turn out to be creeps so sharing a filesystem is a little scary as well.

The “solution” to the creep problem is to not keep sensitive information on Demography computers. You have already promised not to keep data covered by SB 1386², and goes without saying that files that can tie you to illegal activities are also a no-no. For those few files where privacy is an issue (e.g. email) Every file and directory has an owner and the owner can determine who is allowed to read, write and/or execute each file. See the `chmod` command below for how to change the various file *permissions*.

²See the statement of compliance that you signed before we gave you an account

4 The command interpreter

The command interpreter, or *shell* is the program that runs in each terminal window. It waits for you to type something at the *Unix prompt*, `@:>`, and then does what *it* thinks you meant. The shell we use here is called *tcs*h (pronounced “teesh”). Tcsh is one of several modified versions of the original csh (pronounced “sea shell”).

The most important thing that the shell does for you is to let you give commands to the computer. These include the 12 most important Unix commands (Section 5) as well as commands to launch applications like R, Stata, word processors or spreadsheets³. The shell does several other things for you some are essential, some enhance efficiency and others are just cool.

4.1 Essential stuff

Among the essential features of the shell is a mechanism to communicate with running programs that are not expecting user input or have run amok. This is not all that common, but when it happens you need to be able to get the program’s attention and tell it – generally to drop dead. `CTRL + C` does this.

Below is a list of some handy/essential key combinations that the shell recognizes:

- `CTRL + C` : This sends a “kill” signal to the program running in the xterm window where the `CTRL + C` is executed. The “kill” signal generally stops your program – but not always.
- `CTRL + Z` : This “suspends” your program for later restarting. This is a very handy thing to be able to do, but it is extremely confusing if you do it accidentally. the “fg” command brings your program back to foreground.
- `CTRL + D` : This is the End Of File character. Programs which read from the standard input expect a `CTRL + D` to tell them when you are done talking. Many Unix utilities need this explicitly. Sometimes, `CTRL + D` will have a similar function as `CTRL + C` – it is certainly something to type when you are desperate.
- `CTRL + S` :This is the Xoff character. It tells the terminal to stop accepting input. DOS/NT and Unix machines all understand this

³It is of course possible to launch most applications via the menu system or by clicking on corresponding data files in the file manager, but the command line is often faster

character, but new users frequently do not. If you type `CTRL+S` inadvertently, the screen or xterm will freeze until you hit `CTRL+Q`. It is useful if your program is spewing lots and lots of stuff on the screen too quickly for you to read. But mostly it is just a little trap into which users fall. **NOTE:** `CTRL+S` has a completely different and much more useful function in emacs.

- `CTRL+Q` This is the Xon character, it tells the terminal to accept input once again. Try it when the xterm window freezes — just in case you have recently inadvertently hit a `CTRL+S`.

4.2 Efficient stuff

To make you more efficient, the shell offers three particularly nice features: “history”, “`TAB` completion” and “scripting”.

history The history feature allows you to recall **and edit** any command that you have previously issued. To make the **previous** command appear at the `@:>` hit `CTRL+P`⁴. To see even more previous stuff type `CTRL+P` more times. `CTRL+N` will make the **next** command appear — obviously, this makes no sense unless you have type `CTRL+P` at least once.

You can operate on a recalled command using several standard **emacs** editing keys:

- `CTRL+A` To go to the beginning of the current line
- `CTRL+E` To go to the end of the current line
- `CTRL+F` To go forward one character
- `CTRL+B` To go backward one character

The `DELETE` and `BACKSPACE` keys do what you would expect.

TAB completion If you hit the `TAB` key anytime while constructing a command, tcsh will do its best to figure out what you *are planning to type next*. If you are typing a command it will try to find a command that starts out with what you have already typed. If you are typing the name of a file tcsh will try to complete it for you. If what you have typed does not uniquely

⁴the up arrow key will also work

determine a command or filename, tcsh will beep at you and provide a list of possible completions. You can then type a few more characters and hit `TAB` again.

scripting Whenever you find yourself typing the same command several times, it's time to consider scripting. A shell script is just a file of commands that you *could have* entered at the keyboard, but typed into a file instead. You can then set the file's execute bit (Section 7) and execute that file – perhaps now, perhaps later. Using emacs to create that shell script can save you lots of time and hair loss – particularly if many of the commands you are typing are quite similar.

Scripts are also very useful for people who like the idea of being able to reproduce results.

cool stuff The shell is also responsible for displaying the results of the ls command (See 1) in lots of colors.

Regular expressions are combinations of symbols that the shell interprets in clever ways. Generally we use regular expressions to specify lists of files or directories on which a command should operate, but they have many other uses as well. A typical use would be to delete from your current working directory all of the .pdf files whose name begins with a vowel:

```
! @:> rm [AEIOUaeiou].pdf
```

regular expressions come up in several of the “12” important commands.

5 the 12 most important Unix commands

Below is a list of the 12 most important Unix commands. They are simple enough to be easily memorized by anyone who can keep the names of all twelve months in his head.

For the most part, these commands are for logging on and off; for printing; and for moving files and directories around. Many of these commands functions can be done using a file manager or under emacs, but knowing how to do them from the command line, makes you more efficient, reduces errors and opens the possibility of automating tasks with *shell scripts*.

NOTE the `<angle brackets>` indicate that a command argument is *optional* you do not type the `<>`'s it's just a typographical convention

1. `ls <-lgdaF> <regular expression>` The “list file” command, `ls` is used –not surprisingly– to list the names and pertinent information about some or all of the files in a particular directory. The most common and useful option is `-l` that’s a lower case L not a one. It reveals the most interesting properties of your files.

A *regular expression* is a set of special characters (or “meta-characters”) that can be used to represent a list of files or more generally, the set of all character strings with certain characteristics. Regular expressions come up in a lot of places and can be very useful when working with datasets that are full text and/or errors. Regular expressions are used with `ls` to limit the number of files displayed. For example `ls X*Y` would list only the filenames that start with an X and end with a Y; `ls [aeiou]*` would show you all the files that begin with a lower case vowel (In the previous example the `[]` are part of the command)

2. `cd <directory-name>` The “change directory” command makes another directory your present working directory. With no argument, it “moves you” to your home directory. To move one directory “higher” use `..` (two dots) in place of the directory’s name. The parent directory of the current directory is always addressable as `..`.
3. `cp <-R> source file/directory target file/directory` The copy command, `cp` is used to copy files and directories. The `-R` flag is used to copy entire directories and the contents thereof.
4. `mkdir <new-directory-name>` The “make directory” command is used to create a new sub-directory of the current working directory.
5. `rm file-name` or `<regular expression>` The “remove” command is used to remove, or erase files. Here again, regular expressions can be very useful – and quite dangerous.
`rm` is aliased to `rm -i` so that it asks you to verify that you really want to remove a file. If you get tired of this safety feature, use `\rm` instead.
6. `rmdir directory-name` the “remove directory” command removes directories. In order to remove a directory, it must be empty or files and sub-directories.
7. `chmod <aogu +/- rwx> filename-or-directory` The “change mode” command is used to modify the permissions (or mode) of a file or

directory. Permissions are the characteristic of a file or directory which determine who has what type of access to it. All files and all directories have permissions, only the owner of the file/ directory is permitted to change modes.

The first argument is a string of characters that grant (+) or revoke (-) permission to read(r) write(w) or execute (x) the file or directory. The letters aogu indicate who is to receive or lose the given permission. u=user, g=group, o=other, and a=all. Thus to revoke write permission to all users you would type:

```
chmod a-w filename
```

To grant permission to yourself and the group to write and execute a file you would type:

```
chmod ug+wx filename
```

8. `ln <-s> real-file-name artificial-file-name` The “link” command creates an alternative name for an existing file or directory. This is particularly useful when using data sets that you keep in /data/commons (as you should). Rather than typing /data/commons/userid/datafile to reference your data, a symlink would allow you to type something much shorter.

```
ln -s /data/commons/userid DATA
```

would create a link in your current directory called “DATA”. But DATA is really just a secret back way to /data/commons/userid. typing `ls DATA` for example is the equivalent of typing `ls /data/commons/userid`.

It would be a good idea to create the above link right now. Use the `mkdir` to create a new directory in /data/commons called your userid. Then create a link in your home directory so that you can start storing and accessing huge data sets right away.

9. `mv file-name new-file-name` The “move” command changes the name or location within the filesystem of a file or a directory.
10. `less file-name` Variant of the `more` command – less is used to scroll through a file on the screen. While displaying a file, `ENTER` scrolls one additional line; `SPACE` scrolls one additional screen full; `B` scrolls backwards, `Q` quits, `/WORD` searches forward for “word”, `?WORD` searches backward for “word”.

11. `lpr <-Pprinter-name> filename` The “line printer” command prints a file to the named printer.

Printer	Location	Type
age	Basement Lab	HP Laserjet 4015 postscript monochrom duplex 1200dpi
parity	Room 101b	HP 4050n postscript monochrom 600dpi
class	2224 2nd floor	HP 4100n postscript 600dpi
cohort	2232 2nd floor	HP 4200n postscript 600dpi
status	Library	HP 4200n postscript monochrom 600dpi
ses	Xerox Room	Ricoh postscript color duplex 1200dpi

Note that by default each of the above printers is configured to print in economy mode thus saving toner and by extension the world. If you need high quality printouts add “HQ” to the printer name e.g. `lpr -PsesHQ filename`.

12. `pwd` “present working directory” tells you where you are, that is, it tells you which directory the shell thinks is the current directory.
12. `du <directory>` The “disk use” command is designed to tell you how much disk space each directory is consuming. It’s main use, however, is simply to display the directory structure.
12. `man -k key-word — command-name` The “manual” command is used to display manual pages on your screen. To say that man pages are not particularly easy to read is an understatement of almost biblical magnitude. But they are very handy for refreshing your memory or searching for something very specialized.

The man program puts the contents of the man page in a “less” process, see item 10 for a description of how to navigate in less.

In this century a very good source of information on Unix is google. The web knows all about Unix and while there are lots of different distributions, command line tools in particular are nearly identical in all distributions including Solaris, Hpux, every flavor Linux, BSD, and even Apple’s OS X (which by the way is Unix too).

12. `ssh <-l userid> hostname` “ssh” stands for “secure shell” it is really a separate application command is really an application but it behaves like a shell command and is really useful so it is included here. If you type `ssh coale` at the unix prompt, (and then your password when prompted) a remote shell will open on an entirely different machine from the one you are sitting in front of.

The new remote shell on coale will have a prompt like:

`[userid@coale ~]$` indicating that the commands that you type will be executed the machine `coale` which happens to live in the basement of 2224 Piedmont. Happily then new shell will see the same filesystem and understand the same Unix commands.

The reason for ssh'ing to coale is that it is much more powerful than any of the workstations.

NOTE: when using ssh or ssh-like programs on machines outside of the Demography Lab, you will need to specify both your userid (with the `-l` flag) and the fully qualified hostname e.g.

`coale.demog.berkeley.edu`⁵

12. `exit` or `logout` closes the current Unix window, and logs you off – if the current window is the console window.

6 Special and “meta” characters

In addition to the key combinations and commands discussed Unix also supports several characters with special meanings to the shell. Below is a list some of the more common ones:

- * The asterisk or “star” character is used in regular expressions (See item 1). When the shell sees a `*` by itself as in `@:> ls *` it replaces `*` with a list of all the files and subdirectories in the current directory. `@:> ls *` tells the shell to run the `ls` command on each and every file and subdirectory in the current directory. So where `@:> ls` will show files and subdirectories `@:> ls *` will list the files that live *in subdirectories* of the current directory as well.

⁵surprise - from outside the department you will probably end up on malthus rather than coale if you type this. The reason is that from non Demography Lab machines, all `.demog.berkeley.edu` hostnames “resolve” to the outside interface of our firewall. You can ssh to coale from whichever host you wind up on

& The ampersand tells the shell to run the process in the “background”.

When a process is launched in the background, the xterm (See 2) immediately returns with a prompt. When you run a process in the foreground (the usual case) the prompt comes back only when the process exits.

NOTE it only makes sense to run programs in the background if the program spawns a new window. So `emacs`, `stata`, `userfirefox`, or `oowriter` are all fine running in the background. The 12 most important Unix commands are not. They all write their responses to the terminal window. If you put them in the background they cannot do this.

REALLY important: `R` should not be run in the background for the same reason: it runs in the window from which it was launched. This will all make sense after the first week or two of 213.

To bring a backgrounded program to the foreground, type

| `@:> fg %n`

where `%n` is the percent sign followed by a number indicating which backgrounded process you want to foreground. You only need to enter the `%n` if you have more than one process running in the background. Type

| `@:> jobs`

to get a list of backgrounded processes associated with the current xterm.

`~` the tilde character is interpreted by the shell to mean “home directory” by itself, it means **your** home directory, if it is followed by a username as in `~carlm` it refers to that user’s home directory. The `~` can be used in complicated pathnames such as `~carlm/public_html/213F97/welcome.html`. For it to make sense, the `~` must be the first character (and perhaps the only character) of a pathname.

| The “pipe” is used to send the standard output of one process into the standard input of another. For example, if you wanted to know the number of lines in every data file in the current directory you might type: `@:> ls *.data | wc -l`. The `ls *.data` produces a

list of files in the current directory that end in “.data”, the | then feeds this list to the **w**ord **c**ount **c**ommand “wc”. The -l argument tells wc to only report the number of lines. This example assumes that you have named all of your data files *somethingorother.data*.

> The right angle bracket (or greater than sign) is used to send the output of a process into a file. `@:> ls > file.list` would produce a file called file.list containing (surprise) a list of files. Use double angle brackets to append a process’s output to an existing file.

I `@:> ls ~/public.html >> file.list`

would add the names of the file’s in your public.html directory.