# A SHORT UNIX PRIMER

Carl Heiles
January 4, 2011

## Contents



## 1. DIRECTORIES AND SUBDIRECTORIES

For experienced programmers the hardest thing about computing is keeping things organized. You end up with thousands of files and need to keep them in functional categories. You do this by creating appropriate directories and keeping your files there.

### 1.1. The Directory Tree

For example, when you begin a project, such as an experiment, you should create a directory for that project—and that directory needs an appropriate name. For the lab class, you might name the directory `lab1`. For this project you will have several activities: taking data, analyzing data, and writing the lab report. So under the `lab1` directory you might create three subdirectories entitled `data`, `idl`, and `tex`. You have created a little directory tree!

This tree lives under your main directory, called your `home` directory. The name of your home directory is your login name, which is (probably) the first letter of your first name plus the first seven letters of your last name. Your home directory has, as its root, a system directory called `home`. Thus, `home` is the origin of all users' directories. Your personal tree has, as its origin, your home directory. When you create the lab1 directory under your home directory, you've created a new branch of your directory tree.

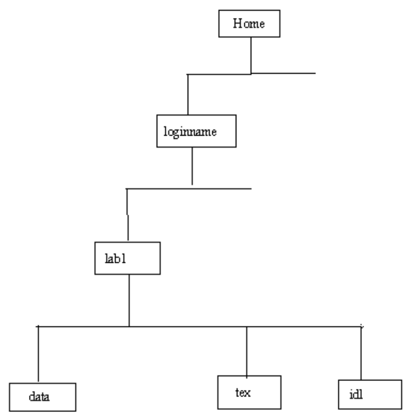


Fig. 1.— The directory tree described in the text.

Figure 1 shows the schematic structure of this directory tree. It is impossible to overemphasize the importance of your creating and maintaining a sensible directory tree.

### 1.2. Working with Directories

We have configured our system so that the prompt contains the name of the current directory. Another way to determine where you are is with the Linux/UNIX command `pwd` (short for "Present Working Directory"). Try it!

The most important commands relevant to directories include:

1. To see a listing of the files and subdirectories of the present working directory (pwd), use the `ls` command. It has some useful options. One of my favorite combinations of options is `ls -lrt`: the `l` means "long listing format", the `t` means "time order", and the `r` means "reverse order"—so this lists all files in reverse time order so that the most recent file appears as the last line of the list. You can also use the `*` character as a wildcard: for example, `ls *ps` lists all files with the suffix `ps` (these are usually PostScript files).

2. To change to a directory, e.g. the `tex` directory under `lab1`, type `cd /home/loginname/lab1/tex`.

Or, shorter: `~` stands for your home directory, so you can type `cd ~/lab1/tex`. Or, if your present working directory (pwd) is `lab1`, you can type just `cd tex`.

3. To create a new directory, e.g. a directory called `monday` under `data`, type `mkdir ~/lab1/data/monday`. Or, if your pwd is `~/lab1/data`, you need only type `mkdir monday`. If you want to eliminate (remove) a directory, get rid of all the files and type `rmdir monday`.

4. To copy a file from one directory to another, e.g. the file `carl` from `~` to `~/lab1/idl`, type `cp ~/carl ~/lab1/idl`. You end up with two copies of the file. You can instead move the file using `mv`. To remove a file, use `rm`.

   We have defined the following options for these commands:

   (a) For `cp`: `cp -ip`. The `i` means "inquire before overwriting a file of the same name"; once you overwrite the original version, it's gone! the `p` means "preserve the original time information about the file"; otherwise, it would tag the copied file with the current time.

   (b) For `mv`: `mv -i`. As above. inquire!

   (c) For `rm`: `rm -i`. Ask to confirm your intention to eliminate the file. Once it's gone, it's gone—there's no "wastebasket".

### 1.3. Permissions for Directories and Files

Permissions determine who has access. You might want your data to be accessible by everybody, for reading, but you probably don't want other people writing over your data! And you shouldn't want your lab writeup to be accessible by anybody, either reading or writing—you don't want to facilitate plagiarism! And it makes sense to keep all your love letters in a separate directory that isn't accessible in any way by anybody else—including their recipient(s)!

You set permissions with `chmod`. Permissions recognize three classes of people: `u` (*user*—yourself!), `g` (*group*—that's usually all the users of `ugastro`), and `o` (*other*—everybody else, *including* somebody in Timbuktu who happens to crash into our system). Note that *group* is essentially like *other*—almost everybody! Each class can have three permissions: `r` (read permission), `w` (write permission), and `x` (execute permission). `chmod` allows you to add, take away, and set exactly permissions for different users with the operators (`+, -, =`).

Suppose, for your data subdirectory, you want read permission for everyone and write permission only for yourself. To grant the read permissions to *group* and *other*, get into the directory above `data` (that's `lab1`) and then type `chmod go+rx data`; to eliminate the write permissions, `chmod go-w data`. To check your work, do a long listing of the directory (`ls -l data`). On your screen, it would write

```
drwxr-xr-x 1 heiles bin 10 2007-01-20 17:58 data/
```

Translation: The first character, `d`, means it's a directory. The next three specify the permissions for the user (that's you): `rw` means you have read, write privileges, and the `x` means you can access the directory. The next set of three characters `r-x` is for the group (all your classmates); the final set of three characters `r-x` means that everybody can read and access the directory contents, but can't write into it.

For your love letter directory (called `love`, you'd want `chmod go-rwx love`

Permissions apply slightly differently to directories and files:

1. *Directories.* In order to access any file in a directory—even to obtain a listing of the files with the `ls` command—the person needs *execute* permission. The read and write permissions are as you'd expect.

2. *Files.* Most files don't need execute permission. By default, when you create a file, it has read/write permissions for you and read for the other two classes.

## 2. OTHER ASPECTS. . .

### 2.1. Command-Line Editing in Linux/UNIX, Emacs, and IDL

One of the joys of Linux/UNIX is command-line editing using keystrokes. We have configured IDL to use the same keystroke commands; Emacs already does so. The most important command-line editing commands are:

| | |
|---|---|
| **arrow keys** | move the cursor as you'd expect. |
| **Ctrl-d** | deletes the character under the cursor. |
| **Backspace** | deletes the character behind the cursor. |
| **Ctrl-e** | moves the cursor to the end of the line. |
| **Ctrl-a** | moves the cursor to the beginning of the line. |
| **Ctrl-k** | deletes the the rest of the line. |

Sometimes, when command-line editing, you inadvertently hit **Ctrl-s**; this prevents the cursor from responding to your keystrokes. If you encounter this condition, type **Ctrl-q**, after which things will work normally again.

In X windows, you can customize any X window to your desires (e.g. fontsize) by putting the cursor on the window, holding down the CTRL or SHIFT key and, simultaneously, holding down a mouse button; each one provides different options. If you are using KDE or Gnome, the details differ.

## 2.2. Aliases

Aliases are shortcuts that you can use in place of typing out a long command over and over again. You can define an alias on the command line; alternatively, if you want to define it permanently, you can define it by editing your `.cshrc` or `.aliasfile` file (which reside in your home directory).

Here's an example. Suppose that you want to force UNIX to check whether it will overwrite a file when you use the `mv` command and, also, to ask you about it. To do this, you use the `-i` option, and you redefine the command `mv` by typing

```
alias mv "mv -i"
```

If you type it in a window, it will apply henceforth to that window alone.

We have included this definition in your `.aliasfile`, which resides in your home directory and is automatically invoked whenever you open a new terminal window—because your `.cshrc` file invokes your `.aliasfile`.

You can check the definition of the alias for `mv` by typing

```
which mv
```

Finally, you can bypass any defined alias by using a backslash. For example, `\rm carl` invokes the `rm` command without the `-i` option that is defined in your `.aliasfile`, which means... it removes the file `carl` *without asking*. Using `rm`, `cp`, or `mv` without the `-i` option is *dangerous*: Once a file has been overwritten or deleted, it's gone for good!

## 2.3. Piping, etc: $|, >, <$

Piping (`|`) directs the output of a command to the next succeeding command. For example,

```
ls | grep /
```

directs the output of the listing command to *grep*, which here selects all names containing the string "/"; those are directories, so this gives a list of directories just under the current directory.

Normally the result of a UNIX command is written to the terminal for you to see. However, you can direct the output elsewhere. For example,

```
more love1
```

prints the file `love1` on your screen, while

```
more love1 > love2
```

creates the file `love2` and writes the content into it.

Normally the input to a UNIX command is expected to be from the terminal. However, you can get the input from elsewhere. For example,

```
mail heiles < complaint.txt
```

uses the file `complaint.txt` as input to the command `mail`, which means that it mails the file `complaint.txt` to heiles; try it! (Do you think he pays any attention to his mail? Do you think it will do any good???)

## 2.4. Remote Logging In

You can log in from home *if* your computer has the secure login software, called `ssh`. If you have Linux, type `ssh ugastro.berkeley.edu`

If you're logging in from home on an (ugh!) Windows machine, you need to be able to use X windows; for this, run the program `exceed` before logging in. You can get such software from the CD *Connecting @ Berkeley*, available for free from the big U.

## 3. COMMON COMMANDS

| | |
|---|---|
| *man commandname* | Gets voluminous (usually overly so) info for you on a specific command. |
| *apropos topic* | Tells which commands are relevant to the *topic*. |
| *pwd* | Shows your “present working directory”. |
| *cd dirname* | Moves you into the subdirectory, below your present directory. |
| *cd ..* | Moves you out of a subdirectory into the directory above it. |
| *cd -* | Moves you to previous directory you were in. |
| *mkdir dirname* | Creates a subdirectory named *dirname*. |
| *rmdir dirname* | Removes a subdirectory named *dirname*. |
| *rm filename* | Removes a file named *filename*; it must be empty. |
| *cp file1 file2* | Copies the contents of *file1* into *file2*. You are left with two files. |
| *mv oldfile newfile* | Moves (or renames) *oldfile* as *newfile*. |
| *cat file1 file2 > fileboth* | Concatenates *file1* and *file2*, writing them into the new *fileboth*. |
| *which cp* | Tells the current definition of *cp* (*which* works for any command); |
| *history* | Gives a numbered list of the previous commands you've typed; typing **!number** repeats that command. |
| *!!* | Repeats the previous UNIX command. |
| *find dirname -name filename* | finds all files with *filename* in and under the directory with *dirname*. |
| *find dirname -name '*love*'* | Finds all files whose names contain the string “love”. |
| *ls -lrt* | Lists the files and subdirectories in the present directory. The *-lrt* gives a long format in reverse time order. |
| *ls -lrt \| grep /* | Pipes the output to *grep*, which selects only those names containing “/” (which are directories). |
| *du -h dirname* | Tells the disk space used by everything in dirname. The “h” means in “human units”. Also handy for giving the directory tree structure. |
| *df -k* | Tells kilobytes used and available on all disks. (“-h” works here too!) |
| *grep -il text file* | Searches the *file* for occurrences of the string *text* The *-i* ignores capitalization and *l* lists only the filename. |
| *lp filename* | Prints the contents of *filename*. |
| *enscript filename* | Fancier print than *lp*. |
| *less filename* | Shows you the contents of the file named *filename* one screen at a time; more flexible than *m*ore. |
| *tail -40 filename* | Shows you the last 40 lines of the file *filename*. |
| *lpq* | Displays the print queue. |
| *cancel jobnum* | Removes the *jobnum* in the print queue. You must own the job |
| *top* | Shows CPU usage, etc, for jobs on your machine. |
| *ps -u username* | List the programs that *username* is currently running on the machine you are logged onto. |
| *kill processnum* | Kills the process listed with *processnum*. You must own the process |