

# Unix Operation Hints

Peter D. Hiscocks  
Department of Computer and Electrical Engineering  
Ryerson Polytechnic University  
350 Victoria Street, Toronto, M5B 2K3, Canada  
phiscock@ee.ryerson.ca  
September 15, 1999

Using a Unix (or Linux) operating system can be speeded up with the hints listed below. These are tips that I learned one at a time by looking over other people's shoulders. If you have some additions to this list, please send them on to me for incorporation.

I'm assuming that you're operating in the X Windows environment and using the Bash shell.

**File Name Completion** Long file names are desirable in indicating their contents, but they are murder to type in. If the file exists in the current directory, type in the first few letters of the name, hit the **TAB** key, and Bash will complete the name. If there is a *name collision*, that is, another file by the same name, it will complete the name up to the point where it needs more input from the human operator.

**Command Completion** Some Unix commands and their options, are long, difficult to remember, and tedious to type in. (*long, difficult to remember, and tedious to type in* are the Unix equivalent of *mad, bad and dangerous to know*). If you have previously executed the command and want to retype it, you can type **!** followed by the first few characters of the command and Bash will complete the command. Of course, this has to be in the same Xterm where you last executed that command. So something like `dvips -f < foobar.dvi >foobar.ps` can be called up again by typing `!dvi`. This is a lifesaver when you are repeatedly processing and viewing a document, as happens when writing something in  $\text{\LaTeX}$ .

**Repeating a Command** Another important hint when repeatedly processing a document: the **up-arrow** cursor key causes the previous command to be listed at the cursor. Hit **return** and it's executed again.

Each press of the **uparrow** cursor key takes you back one command in the *command history* list, so if you want to execute some previous command, you just keep pressing **uparrow** until you see that command again and press return. This is Very Handy indeed.

If the previous command is something similar to the next command you'd like to execute, it's often efficient to cause it to be listed with the **uparrow** cursor key, and then edit that command into the correct form, using the **Backspace**, **Delete** and cursor movement keys. Then press the **return** key to cause the new command to be executed.

**Finding Files** The **find** command isn't the easiest command to use, but it's awfully important in finding files. The syntax for the usual case is `find . -name foobar`. Notice the `.` (Some

versions of the `find` command require you to specify that it should print the name when it finds it. The syntax in this case is `find . -name foobar -print`). In any case, the `find` command tells the `find` command that it should search for a file name *foobar*, starting in the current directory (`.`). The `find` command searches recursively, so it will search all the subdirectories from the current directory. The search process can take a while on a large system, so you might want to run this command in the background by appending an ampersand: `find . -name foobar &`

An alternative technique which I've used extensively on my personal Linux box, is to do a recursive directory search and dump that into a file for future reference. Move to the root directory (`\`) and then execute `ls -R > dirlisting`. This will recursively list directories into a file called *dirlisting*. Then you can use your favourite editor to search through *dirlisting* for a given file. Since you probably know how to tweak the editor search command much better than the idiosyncracies of the `find` command, this is often useful. Moreover, if the directory structure doesn't change much over time, *dirlisting* can be repeatedly used to find files. This can be a lot faster than waiting for some other search command to complete.

**Finding a file containing a certain string** Here's the situation: you wrote a file and you don't know where you put it. But if you *can* remember a word or phrase (such as the ever-memorable *foobar*) that the file contains, then you can find it using the following command:

```
grep foobar `find . -name "*"`
```

This has the effect of doing a recursive `find` operation starting at the current directory (`dot`) for all files. These names are then given to the `grep` command, which looks for the string *foobar*.

Notice that the ``` symbol is the single backquote, not the single forward quote `'`. Thanks to Cenk Bilgen for suggesting this command.

An alternative command, suggested by Jason Naughton, which may be useful if your keyboard is missing the ``` symbol:

```
find . -name "*" -exec grep foobar {} \;
```

In this version, the `find` command does a recursive search for all files, each time executing the `grep` command.

**Cut and Paste** This is *really* useful. You can copy and paste between different Xterminals on the desktop. Using the left mouse button, drag the mouse over some text you wish to copy. It will be highlighted. Then, in another xterm, press the middle mouse button to drop the highlighted text.

You can use this if you're copying and pasting between two documents, for example. Start the source and target text editors in two xterm windows, and then you can highlight and drop text from one into the other. Be careful, though, the only text that gets copied is what is visible in the window. If text overlaps the right edge when you highlight, the hidden part will not be copied.

You can also use mouse copy and paste within one document, providing you can see the material you'd like to copy and the point in the document you'd like to copy it to. However, once you've highlighted some text, you can't scroll the document: the highlighting is cleared by a *scroll* operation.

**Moving between Directories** Moving around in a Unix system can require a lot of typing. One trick is to open lots of Xterminal windows, iconizing them when they are not being used to reduce desktop clutter. If you have to look in a number of different directories, each Xterm can show

one directory. Also useful: the command `cd` by itself takes you back to your home directory. The command `cd -` takes you back to the directory you were last in. So repeated `cd -` commands enable you to alternate back and forth between two different directories.

**Copying files between Directories** It is a great time saver to use an existing file as the skeleton for a new document. For example, an existing  $\text{\LaTeX}$  document can be copied, renamed and then edited to create a new document, removing the necessity for retyping and for remembering all the formatting commands.

Here is the trick for copying a file *foobar* from one directory to another:

- Open one Xterm in the target directory and another in the source directory.
- In the target directory, execute the `pwd` (print working directory) command to get the full path name of that directory.
- In the source directory, type in `cp foobar` and then use the mouse copy-paste operation to copy the target directory path name that resulted from the `pwd` command. The pasting operation in the source directory will put the correct path name at the end of the copy command. Press **return** and the file will be copied, as you can verify with the `ls` command in the target directory.

**Processing a Bunch O' Files** Suppose you have a bunch of files in a directory that you want to print or convert to another form. For example, you might have a dozen postscript files in a certain directory and you would like to generate the `.pdf` format. Rather than type in a separate command to process each one, which is tedious, you can type in the shell script from the command line that accomplishes this. Here's an example screen shot showing how you print some postscript files, that illustrates the technique: (the `phiscock@localhost:` and `>` symbols are prompts from the machine).

```
phiscock@localhost: for i in *.ps
> do
> echo $i
> lpr -Poffice $i
> done
```

The first line of this script sets up a repeditive loop with the variable `i` being assigned, one after another, to all the postscript files in the directory.

The `echo` command on line 3 writes each file name to the screen so we can monitor the progress of the script. (If you see it printing a file you don't want, hit `<control>c` and the script will stop.)

Then, in the fourth line, each name is passed to the `lpr` command for printing on the office printer.

To convert a bunch of postscript files to `.pdf` format, the incantation would be similar:

```
phiscock@localhost $for i in *.ps
> do
> echo $i
> ps2pdf $i
```

> done

where `ps2pdf` generates the new `.pdf` file each time.

This sort of thing can save a *lot* of work.

**Acknowledgements** Special thanks to Jason Naughton, Cenk Bilgen and Ken Clowes who identified a number of these techniques.