More Unix Commands

Written by Dave Pawlowski

Introduction

At this point, you should be comfortable with navigating the Linux filesystem, have cursory knowledge of the hierarchy, and know how to create and remove files and directories. Now it's time to add more commands to your tool belt so that you can start doing something useful.

more

You've learned how to quickly display the contents of a file using cat. But what if the file is huge?

Try:

```
% cat ~dpawlows/Documents/Classes/Phy379/Notes/linuxsystem.tex
```

You should get a lot of text printed to the screen. Now you have to use the mouse wheel (if that even works) to scroll up through the text.

An alternative is to use more:

```
% more ~dpawlows/Documents/Classes/Phy379/Notes/linuxsystem.tex
```

Now you should see the first "page" of text, and you can advance through it using the space bar or the page up/page down keys. Pressing q will allow you to exit without needing to go all the way to the bottom.

head/tail

Maybe you don't need to see the whole text, but you just want to see the first or last few lines. Then you'll want to use the head or tail commands

```
% head ~dpawlows/Documents/Classes/Phy379/Notes/linuxsystem.tex
```

will show you the first 10 lines of the linuxsystem.tex file, while

```
% tail ~dpawlows/Documents/Classes/Phy379/Notes/linuxsystem.tex
```

will show you the last 10 lines. Adding the -n M flag will show M number of lines instead of the default 10.

Pipes

You've learned about how to redirect the output of one command into a file using cat. It is also possible to redirect the output of one command into another command. This is called piping.

```
% cd /usr/bin
% ls | more
```

will cause the output of ls to be displayed in the style of more providing you with a list that you can scroll through page by page.

Piping is an extremely powerful feature of Linux, and this simple example doesn't do the command justice. A common use of this, especially on large computer systems, is to use it with the ps command. ps tells you about the processes that are running on the computer. Typically, this can be a very long list, but you can extract useful information from it if you combine it with another command.

grep

grep is an extremely useful command that searches a file or files for text. For example:

```
% cd ~dpawlows/Documents/Classes/Phy379/Notes
% grep -Hi code *
```

tells Linux to search all the files in the above directory for the word 'code'. You should get several lines of text, all of which include that word. The -H flag tells grep to return the file that the word appears in and the -i flag causes grep to ignore case sensitivity.

grep is obviously useful for searching files, but it can also be useful attached to other commands:

```
% ps aux | grep -Hi 'username'
```

will tell you information about all the processes that are being run by you on this computer. There probably aren't many. If you replace your username by mine, dpawlows, you should see a few more.

ps

The ps command is useful when you want to see what processes are eating up compu-

tational resources, and if you need to kill an unresponsive process, provided you are the owner of the particular process. This is done using the kill command:

```
% kill 'pid'
```

where 'pid' is the process id number, the number in the 2nd column of the ps output.

For example, try

```
% emacs -nw &
```

Hit enter a couple times after that to bring the prompt back. You've just put the program emacs in the background. Lets say you can't get it back into the foreground in order to close it. You can use ps and kill:

```
% ps aux | grep 'username'
```

note the pid of the emacs process.

```
% kill 'pid'
```

You will need this because sooner or later, you will start a process and put it in the background, forget about it and log out of chuck. Since the process is in the background, it will not necessarily end. Later on, you can find it again and kill it using this command.

Finding files

The most useful command for finding any file on your system is the find command. find has a ton of options which may be useful, but the best uses are the following:

```
% find . -name 'string' -print
```

will search the current directory for a file called *string*. You can include wildcards in both the directory (replace . with any directory), and in the string (if you include the quotes). For example,

```
% cd ~dpawlows/UpperAtmosphere
% find * -name 'champ*' -print
```

will search the UpperAtmosphere directory for files starting with the string champ.

If you include the grep command, find can find a file anywhere in the system that contains a string *inside* the file.

```
% cd ~dpawlows/idl
% find * -exec grep -Hi 'champ' '{}' \; -print
```

will output the lines and files that include the word 'champ' in them. The syntax is complicated, but necessary. Note that this is useful if you really have no idea where the file is, but remember a specific word or phrase contained in it.

Depending on how you use this, it can take a very long time and search a lot of stuff. For this reason, the locate command can be better, if it is available.

linking files

It is often useful to link two files together. For example, try:

```
% cd ~
% ln -s ~dpawlows/Public/Phy379/ Phy379
```

This will create a soft link in your home directory that points to the Public Phy379 folder. Now, if you want to cd into that directory, you just have to do

```
% cd ~/Phy379
```

and if you want to copy a file to that directory or a subdirectory of that directory, its quick and easy

```
% cp homework1 ~/Phy379
```

This works even though the folder that Phy379 is linked to isn't your folder.

Editing files

So far, you have been able to create and view files using cat, but you haven't been able to actually edit them. We could have an entire course on text editors but we will cover the basics in this course over the span of the next few tutorials.

There are a variety of text editors that are typically installed on most Linux systems. Of these, emacs and vi are installed on nearly every system. These are known as WYSIWYG (what you see is what you get) text editors. They are **NOT** Microsoft Word, in which the program is always getting in the way of what you are trying to do, changing spelling, altering the formatting, etc. Emacs and vi provide a powerful tool to create text files, the bread and butter of what every programmer does.

Since I use emacs, you will learn emacs. Nuclear wars have nearly been started over which text editor is better. Chances are which ever you start with, you will learn to love.

Note that both of these editors were designed to be run in a terminal window, not a fancy X-window. This means that they were designed to be controlled using only the keyboard. Though this has changed with the advent of those mice things, using the keyboard to do everything (open, save, move, close, highlight, etc). is the most effective and efficient way to use emacs and vi. This makes them a pain to learn, but well worth it in the long run.

You will get to know emacs in the next tutorial, but you are welcome to try it on your own. You can create a new file by entering

```
% emacs -nw newfile
```

which will open the file 'newfile' in emacs in a terminal window. You'll want to checkout http://www.stanford.edu/class/cs107/other/emacsrefcard.pdf and get a copy of the emacs reference card for help with the keyboard shortcuts if you get a head start.