

Lab 0

25 August 2015

In this course, the two main tools we'll use are a compiler (`g++`) and an editor (`vim`), and we will use them under a particular operating system (Linux). Those tools aren't the primary focus of the course, but like a chemistry course that needs to teach you how to use the Bunsen burner or an art class that needs to teach you how to clean the paint brushes, I need to spend a bit of time up front teaching you the equipment so that we can get on to the main event. The compiler will show up next week; for now we'll get a first look at the other tools.

The very first order of business is getting logged in to the department's computer systems. I've created for each of you a remote account on our machines. It is entirely distinct from your LancerNet account; please read the Acceptable Use Policy while I'm handing out your individual account information. Once you've read the policy and have your account information, you're ready to log in.

I'll walk the whole class through this at once, but the extremely brief version is: you will use the program called "PuTTY", to connect to a server named `cs.longwood.edu` with a port number between 220 and 244. Once connected, you'll type in your username and password, and then you'll change your password. A longer version of these instructions is linked from the course webpage, in case you forget, with some additional links to let you do this from your own computer later.

Getting started on the command line

You should now have a window open with a connection to one of the CMSC machines; it is a text-only connection, with a line or two of login information and then a prompt. This prompt, and by extension this kind of text-only window, is known as a "command line". You interact with it by typing commands, and it responds by giving you (text) results of those commands.

For instance, type

`w`

and hit enter. The `w` command gives a listing of everyone who is logged into

the system you're on and some information about their session.

As in more graphical systems, some commands require extra information, but rather than popping up a dialog box to request additional input, you provide it up front. The `finger` command gives some information about another user, and requires you to provide a username; if you type

```
finger dblaheta
```

you would see my information, but if you replace `dblaheta` with your own username, it would show your info instead. (Try it!) In fact, you can also try it with first or last names, and it will print out all accounts that match.

When we talk about instructions like this, we often refer to the first part (here, `finger`) as the command itself, and everything else (here, `dblaheta` or whatever else you type) as the “command line argument(s)”.

In this course, essentially all your interactions with our department machines will be through the command line. I'll introduce you to other useful commands as the need arises, but feel free to investigate on your own as well. There's a lot out there!

Files and directories

As on other systems, your files can be organised into directories (which may themselves contain other directories). When you start, your working directory is the one known as your “home directory”; whenever you specify a file, it's assumed to be in the current working directory, which means that one of the first things you need to learn to do is look at directories and move between them.

To get started, type

```
ls
```

(that's a lowercase L, not a one) and hit enter. This command lists the contents of your home directory—which is currently empty.

Now, type

```
mkdir lab0  
ls
```

and see what changed: there is now a directory for your work on lab 0. Remember, though, that it is not yet your working directory— you are not “in” it yet. Type

```
cd lab0
ls
```

and notice two things: first, the prompt changes to reflect the fact that you are “inside” the `lab0` directory now; and second, when you type `ls`, there are no contents yet, since this is the directory you just made and haven’t put anything in yet.

At this point, read over the “Command line starter kit” at the back of this handout. You should keep that page handy for your own reference, until it becomes a bit more familiar.

Now to create a file.

Editing a file with vim

Just for starters, we’ll use vim to create a new file with some arbitrary stuff in it. Type “`vim sample.txt`” (or use any other filename that catches your fancy) and hit enter. The filename is the command-line argument, and if the named file doesn’t already exist (like right now), it will be created as soon as you tell the editor to save what you were working on.

The main thing you’ll need to remember to learn vim is that at any time you’re either in “command mode” or “insert mode”. Insert mode is roughly like what you’re familiar with from word processors—it lets you type in text—but command mode is different. Command mode is where you do the editing work, and when you’re in command mode, each letter actually executes a command, like “remove this character” or “paste from the clipboard” or “jump to the end of the file”. Vim always starts out in command mode; switch to insert mode by pressing the `i` key. You’ll see that you’re in insert mode because vim will now say `-- INSERT --` on the lower left.

Just to fill space, type two or three lines of text. Any text. Be creative.

When you’re satisfied with your small masterpiece, you’ll press the Escape key (in the upper left corner of the keyboard) to return to command mode. There are a few things you can do from command mode (aside from returning to insert mode); for instance, you can press `x` to delete a character or `dd`

to delete a line. If you do that but change your mind, you can press `u` to undo (and press it multiple times to undo multiple things). If you undo too much, you can press Control-R to re-do a thing.

Go ahead and use those commands to see how they work! You can also re-enter insert mode to add more text by again pressing the `i` key (and return from there to command mode by pressing Escape).

After you’ve tried all of those, you can write out the file (i.e. save it) and quit the program by typing

```
:wq
```

and hitting Enter.

Also at the back of this lab handout is a summary of this section. I want you to keep that handy, too—until you fully internalise how vim works, you’ll need to refer to the sheet to remember what you can do. Vim takes a bit of getting used to. However, it is popular among technical people who are willing to invest the time to learn its many tricks, because it rewards that investment: once you learn vim’s controls, it makes most standard editing tasks *much* faster than if you were simply using the arrows and spacebar and delete key. Try to develop the mindset of being in command mode most of the time, and only switch to insert mode to type something new, then “escape” from it to move around the file and do further editing.

Anyway, now that you’re back on the command line, you can use `ls` to verify that your newly-created file actually exists, and you can type “`cat sample.txt`” (or whatever you called it) to display its contents.

Edit the same file again and add a few lines to it, and save the new version.

Then, in the same directory, make another file called `students.txt`, whose contents are your name and the names of the two people sitting nearest you in the lab (introduce yourself!).

Then, make a third file, called `computer.txt`, whose contents say what kind of computer(s) you have and what system they’re running. (The main purpose of this is really just to give you practice editing files, but the information might come in handy too at some point.)

The last part of this lab is making sure you’re familiar with the handing-in process—it will be more important in future weeks, of course, but for now it’s useful to test out the logistics.

Handing in

Your work for this lab includes multiple files within the `lab0` directory, which need to be bundled together and put in a place I can find them; and there is, as they say, a command for that.

First, though, make sure you are no longer “in” the `lab0` directory. If you type `cd` by itself and hit enter, it will return you to your home directory. Then type

```
handin cmsc160-3 lab0 lab0
```

Note that you need to type `lab0` twice: the first time, it is the name of the assignment, and the second time, it is the name of the directory you’re trying to hand in. (If you had called the directory something other than `lab0`, you’d type that instead as the last command line argument.)

If all goes well, you should get a message to that effect, and it will list all the files in the handin (specifically, the three short text files you wrote). If something is amiss, there will be an error message to tell you what to fix.

Future note

You won’t need this information yet this week, but it may come in handy when you start reading Chapter 2: On this system, you can edit program files (the book will start talking about a file named `hello.cpp`) using Vim as described in this lab. When the book talks about *compiling* the program, you do that with a command called `compile`:

```
compile hello.cpp
```










Then, to *run* the resulting program, you would type the command `./a.out`:

```
./a.out
```

We’ll talk more about this in class next week!

vim starter kit

Eight things you must know to use vim


1. The vim editor is modal. The main two modes are “command” mode and “insert” mode. When you are in insert mode, everything you type goes directly into the file. When you are in command mode, every key you hit is some command, such as “undo” or “delete this character”. You always start out in command mode, so you’ll have to enter insert mode before you can start typing text. Modal editing
2. The escape key ( in the upper-left corner of the keyboard) always gets you back to command mode. If you are already in command mode, it does nothing (except beep). If you don’t know what mode you’re in, hit escape and then you’ll know you’re in command mode. If you get a little lost, you can always hit ESC to get back to a known place. Escape! Command mode: 
3. From command mode, hit the ‘i’ key to enter insert mode. Insert mode: 
4. In command mode, hit the ‘x’ key to delete the character underneath the cursor. Delete char: 
5. In command mode, hit ‘dd’ to delete the whole line the cursor is on. Delete line:  
6. You can undo your mistakes! In command mode, hit ‘u’ to undo the most recent thing you did (and ‘u’ again to undo the thing before that, and so on). Note that a whole insertion, from the time you hit ‘i’ to the time you hit Escape, is considered one action—if you just need to delete a small amount of text, use ‘x’ or ‘dd’. Undo: 
7. In command mode, Control-R will redo the action you just undid. Accidentally pressed ‘u’ and lost an entire insertion of text? Control-R (abbreviated Ctrl-R or just ^R) will “undo the undo” and put the text back where it was. (Whew.) Redo:  
8. If you’re in command mode, type :wq and hit Enter to save the current file and quit the editor. Save and quit: :wq


It’s a little strange at first, but once you get the hang of using a modal editor it’s actually pretty powerful. It gets a *lot* easier with practice.

Command line starter kit

Eight useful tips for the command line

Commands. All commands are executed by typing the name of the command. Many commands also require the names of one or more files or directories; these are separated from the command and from each other by spaces.

Working directory. At the command line you are always “in” some directory; to print the name of the current working directory, type `pwd` and hit .

Home directory. The directory you start in is called the “home directory” (and its full name is usually of the form `/home/yourname`). You can always get back to it by typing `cd` and pressing . (Clicking your heels three times is purely optional.)


Navigating directories. To change to a different directory, type `cd` followed by the name of a directory. The special names `..` (meaning the “parent” or “up one” directory) and `.` (meaning the current working directory) can always be used wherever directory names are expected (although “`cd .`” isn’t very useful).


Modifying directories. To make a new directory, use `mkdir` with the name of the directory to make. To remove an empty directory, use `rmdir` with the name of the directory to remove.

Manipulating files. Four important commands are `cat`, `cp`, `mv`, and `rm`:

<code>cat file1 [file2...]</code>	Display contents of file(s) on the screen
<code>cp filename newname</code>	Copy existing file, with new name
<code>cp file1 [file2...] dirname</code>	Copy existing file(s) into directory
<code>mv oldname newname</code>	Rename file from old filename to new filename
<code>mv file1 [file2...] dirname</code>	Move file(s) to a directory
<code>rm file1 [file2...]</code>	Permanently remove (delete) file(s)

For all the versions that permit multiple filenames, you can use `*` as a filename wildcard, e.g. `rm *.o` to remove all the files ending in `.o`.

Repeating commands. If you need to type a command that is the same as another you’ve recently done, or similar, use the up arrow to see those commands again. You can edit the line if necessary, and then press  to run that command again.

Tab completion. If you press the  key while typing a filename, the command line can sometimes fill in some or all of it for you.