

INTRODUCTION TO UNIX (I)
 Notes by Sean Hogan, last revised 1/4/12
 seanhogan@---.edu

Answer this afterwards!!

<http://tinyurl.com/unixone>

(I) WHAT'S A UNIX?

Wikipedia will help us here - in short, Unix (pronounced: you-knixs) is a timesharing computer operating system developed ages ago (see: 1969). By timesharing, it was a way to allow multiple people to use a single computer back in the 70s. There's a lot more about its history all over the internet, but what's important is that it had a set of ideas that influenced many modern day OSs (Operating Systems, more or less, the software that allows stuff to run on a computer) like Linux and Mac OS X (Called Unix-Like OSs)...

We call this course Introduction to Unix, because it's a nice encompassing title for ideas you'll encounter across programming/interacting with Unix-like systems.

(2) NOW WHAT?

Use the guest login to make your own account:
 Username: introunix
 password: Radish

If you're reading this at the class, you're sitting at a Linux box (computer). Linux is just the name of the operating system, these machines run the Debian distribution of Linux. (You may have heard of other "distros" - Ubuntu comes up often).

If you don't have a UChicago CS account already, you'll need one to log into the Linux boxes:

https://www.cs.uchicago.edu/info/services/account_request

So log in (with the CS ID/password you just made)! Are you logged in? Good. You'll notice that not much looks special about this - it's a GUI that you're sort of used to, either with Mac/Windows experience. You can interact with files, save documents, draw silly pictures, etc. as usual by clicking on folder icons and so forth. But we're here to do more! So open up the Terminal application! (Applications -> Accessories -> Terminal).

(3) YOUR HOME DIRECTORY AND OTHER THINGS

***The words directory and folder, more or less, are interchangeable. I'll try to stick to using directory since it's better that way, but I might slip.

Each of the names has some sort of history to it, but all we need be concerned with is that they're pretty much used exchangeably. If you've done everything right, you should see a black window, with the following text:

```
your-ID@the-computer-you're-on:working-directory$
```

In my case, I have

```
seanhogan@rigel:~$
```

which means I'm logged in as "seanhogan" at the computer called "rigel", and my current working directory (i.e., "what folder I'm in") is "~", or my "home directory". The colon only serves to delimit (separate) the working directory and the name/computer info. Similarly, the dollar sign "\$" delimits the previous 3 pieces of info, and your input (what you type into the terminal window).

You might be able to guess a few commands to type into the window. Namely, "help", which...won't be of much help at this point.

At the moment you're all actually in the same folder so we're going to want to give everyone their own workspaces.

Don't actually use "seanhogan", use your CNET ID, or horrible things will happen.

Enter the following. (Note, lines like this mean you should type in whatever is after the dollar sign.) It will make a directory for you and put you in the directory.

here's what i'd do:

```
$ mkdir seanhogan
$ cd seanhogan
```

Your terminal prompt should read

```
introunix@computer:~/your-id$
```

We'll start with a few basics - "cd" (change directory) and "ls" (list).

```
$ ls
```

You should see nothing. That was uninteresting.

```
$ cd /
```

and you will now be in the "root" directories. Check out what's in here (using "ls"). You probably won't need to worry about what's here for a while. Of interest to us is the "home" directory. Go to it ("cd home"). See what's in here - a bunch of directories, one of which is the ID "introunix", and likely a few of others. Yes, you can look at your friends' documents if you desire! There are ways to prevent this, as we'll discuss later. For now, just go back to your home directory, either by

```
$ cd ~/your-id
or
$ cd /home/introunix/your-od
(where your-id is, well, your id.)
```

Also note that like "~" is your home directory, "." represents your working directory, and ".." represents the directory one level above, if it exists (so doing "cd .." while in "~" will take you to /home) You can even view these "directories" - "." and ".." by executing `ls -a` (list all).

(4) MORE COMMANDS, FILE PERMISSIONS!

So, it turns out we can do a lot more than just look at things in folders. While in your ~/your-id directory, enter

```
$ mkdir -v unix-i
```

which will create a directory in your home folder called "unix-i". The "-v" is called a "flag", and in this case it tells the command (mkdir) to be verbose about what it's doing (in this case, making a directory). In general, commands take the form

```
$ COMMAND [OPTIONAL FLAGS] [ARGUMENTS]
```

where arguments are usually information you give the command - in this case, the directory name. Check out your new fancy directory, and enter it. Unsurprisingly, there's nothing there. Let's remedy that:

```
$ echo "words" > junk1
```

and make another directory:

```
$ mkdir junkdir
```

Don't worry about how the echo command works or the ">" symbol for now, but your directory now has a directory called "junkdir", and a file called "junk1" in it, with the text "words". Great! Now enter

```
$ cat junk1
```

and notice that junk1's contents were displayed. This is useful. With "ls" we can see that the file is there, but what if we want to know more about it? For a "long" listing, enter

```
$ ls -l
```

and you'll see some more information. Here's my output:

```
seanhogan@rigel:~/unix$ ls -l
total 4
-rw-r--r-- 1 seanhogan college 23 2011-09-28 16:46 junk1
drwxr-xr-x 2 seanhogan college 4096 2011-09-28 16:57 junkdir
```

What's important here is the 2nd line starting with "-rw-r--r--". That's the file permissions of the file in question. The first character tells you if we are looking at a file or a directory. A "-" means file, a "d" for directory (actually, directories are files, but don't worry about that for now.). There are 9 more characters, split into 3 groups of 3. Each 3-tuple corresponds to the read, write, execute permissions. The groups from left to right are for the creator, the group, and anyone else, e.g. for junk1, no one can execute (run) junk1, I can write (modify) junk1, and anyone can read ("open") junk1.

It'd be nice to change these permissions. There's a nice way to do this with some binary. Note we can say that

```
rw-r--r-- = (110)(100)(100) = 644.
```

If we're the owner of the file, we can change the permissions. Let's set the file so anyone can execute it, and keep the other permissions the same:

```
$ chmod 755 junk1
```

(alternatively, we could have used "chmod +x junk1")

Now our permissions read "-rwxr-xr-x". Great.

At this point you can read/write protect all of your stuff, although if you're logged in the guest account there's nothing you can really do (You might want to see if your CS account works at some point)

Back to the output line we're concerned with:

```
-rwxr-xr-x 1 seanhogan college 23 2011-09-28 16:46 junk1
```

The 1 isn't important at the moment. The next field is your id, the next field is the group associated with the file - so anyone in that group who isn't you will abide by that file's group permissions. The next field is the size of the file in bytes, and the rest are self-explanatory (when did I write this tutorial originally???)

OTHER NICE THINGS:

-Note that we can also use "ls -ld" or "ls -l -d" and get information about our working directory. Useful.

(5) REMOVING THINGS...and TAB COMPLETION!

Sometimes you need to delete things. In Unix-like systems, things work a bit differently...when you remove something, it's gone. Forever. So be careful! (There are ways around this). Let's delete those files we made. First, stick another directory inside of junkdir. Call it whatever you want. Make sure you're back in the unix directory (use "cd .." if you forgot), and:

```
$ rm junk1
```

Notice, junk1 is gone. Forever! How sad. Be glad that's not your source code for your latest CS lab, or the latest LP you painstakingly "found" on the internet. Let's try to remove junkdir.

```
$ rm junkdir
rm: cannot remove `junkdir': Is a directory
```

Oh no. What do we do now? Type "rm", then hit the tab key twice.

```
seanhogan@rigel:~/unix$ rm
rm      rmail      rmdir      rmid
rmadison  rmcp_ping  rmic      rmiregistry
```

What's this? Nothing short of

TAB COMPLETION!

By pressing tab, the terminal has automatically shown you what possible commands you could mean that start with "rm".

Now, type "rm j" and hit tab. You'll notice your input lengthened automatically to "rm junkdir/". What just happened??

TAB COMPLETION! AGAIN!

REMEMBER IT! It will save you lots of frustration, by showing the possibilities of completing a command or some file you're typing about, or listing whatever directory you want to cd to. I did not learn the magic of tab completion for a few months. And now you know. This will save you from typing absurd things like "cd /usr/lib/hadoop-0.20.203.0", typing the wrong number and having to try again.

TAB COMPLETION! (Don't forget it!)

Right, so back to the output when we tab-completed commands starting with "rm". What could be useful..."rmdir" looks nice. Let's try that.

```
$ rmdir junkdir/  
rmdir: failed to remove `junkdir/': Directory not empty
```

Oh, darn. You can't delete a directory unless it's empty. If you like pain, you can go delete everything inside it yourself. Or...

```
$ rm -rf junkdir
```

You might see a few jokes on "rm -rf". The -r flag tells rm to delete recursively, essentially nuking everything in the folder. -f tells rm to do this, no questions asked. Needless to say, be careful with this command! Everything can get shot right to hell if you do this to the root directory (although you're not allowed to, thanks to permissions. Although if you wanted to, you could do this on your own computer.) On the bright side, it could hypothetically be a great way to instantly delete those "animal photos" you've been keeping stashed on your hard drive.

...

(6) moving and copying

mv & cp (move/rename and copy).

You can move files around with move. the syntax is

```
$ mv [source file(s)] [destination].
```

mv works in a number of ways, so it's easiest to try and figure these things out yourself and I'll show you solutions as well. Mastering this will be useful.

1. Move a file into a directory.
2. Move multiple files into a directory.
3. Move a directory into a directory.
4. Move a directory and any number of files into a directory.
5. Rename a directory.
6. Rename a file.
7. Move *all* files and directories in a directory into my working directory.

Assuming my working directory has two directories, "d1", "d2", each with files "d1-f1","d1-f2", etc. and two files in my working directory, "f1" and "f2", solutions can be:

1. mv f1 d1
- 2a. mv f1 f2 d1
- OR
- 2b. mv f* d1
3. mv d1 d2
4. mv d1 f1 f2 d2

```
5. mv d1 new-d1
6. mv f1 new-f1
7. mv d1/* .
```

A few things to notice: mv renames things, also:

What the heck is the asterisk "*" doing? IT's a "wild-card", and it matches one or more of anything! This is handy when you have thousands of files called "file-0001", "file-0002", etc... that you want to move somewhere else. In the solution 2b, f* matches both f1 and f2, so the mv command moves both of them into the directory d1.

Similarly, in 7, the * matches everything in d1 (in this case, "d1-f1" and "d1-f2"), and moves it to ".", the working directory.

Copy, "cp", works sort of similarly:

```
$ cp [source file] [destination]
```

1. Copy a file.
2. Copy a folder (you'll need a flag! It's -r, for recursive.)

solutions:

1. cp file the-same-file
2. cp -r folder the-same-folder

creates copies of file and folder in your working directory, respectively called "the-same-file" and "the-same-folder".

While this may seem archaic at first, it's very elegant when writing scripts, and is becomes much quicker over time for the majority of uses.

(7) MANPAGES

AND ANOTHER IMPORTANT COMMAND!

"MAN"

```
$ man ls
```

Man can be used with most commands, and gives a "MANual" of the command. This usually includes syntax, as well as all flags you can use. Learn to use this well! You navigate the "manpages" with "b" and the spacebar - "b" goes back a page, space goes forward. likewise you can use pgdown/pgup or the arrowkeys.

(8) Summary, so far, plus other things (mv and cp)

Commands learned -

```
rm cd ls mkdir rmdir chmod mv cp man
```

It's best to play around, looking up new commands as you need to, or reading the manpages. Any good search engine will be very useful in your near future.

After this you should be comfortable with:

Moving, copying, deleting files/directories, basic navigation of the filesystem, understanding some of output of ls -l.

Hopefully with practice, you'll be able to see how much more efficient it is to manipulate files through the filesystem, among other things.

Next time we'll learn a little about signals, compression/archiving, and file transfer, etc, or whatever.