EECS 211 Lab 1

Navigating the Unix Shell

Winter 2017

Today we are going over the basics of how to log into a remote computer, use shell commands to create and edit files, and compile and run C++ code. The Northwestern EECS servers run a Unix shell called *tcsh*.

Using *tcsh* is very similar to using *bash*, the default shell that Macs use for Terminal.app.

The shell works as a textual conversation. It presents a prompt, like `[wsc147@batman eecs211]$`. (The default EECS prompt shows the username, the hostname, and the current working directory.) You type a command and press enter. The shell executes the command and then prints another prompt, waiting for further commands. For example, to list the files in the current directory, you will run the *ls* command by typing it at the prompt:

```
$ ls
```

Don't type the `$`. That stands for your shell prompt.

Before you can do that, though, we have to get you logged in.

Logging in

For the majority of you who are unfamiliar with the Unix shell, it probably seems like a scary foreign concept reserved for computer hackers on TV shows and movies. However, in reality, with a little bit of time and a few basic commands, you will realize that the Unix shell is not something to be scared of, and in fact a very useful tool to embrace as you continue your computer science education. Don't get frustrated if it seems hard at first! Every great computer scientist was at one point also unfamiliar with the shell, just like you, but with a little bit of exposure, it will start to make sense.

SSH (secure shell) is a protocol that allows you to login remotely onto an external system. We will be using it in order to create a connection onto a Northwestern remote server, where we will be learning our first Unix skills. For the first step of establishing the connection, it will be different for Windows and Mac/Linux, but for the rest it should not matter which OS you are on, since you'll be using the remote Unix machine.

Windows

Download the SSH client PuTTY. The link on the right will take you directly to the Windows installer. After you install PuTTY, open it up. You'll need to enter a hostname to login to. The link on the right will take you to a list of student lab hostnames (such as *tlab-*

https://the.earth.li/~sgtatham/putty/latest/x86/putty-0.67-installer.msi

http://www.mccormick.northwestern.edu/eecs/documents/current-students/student-lab-hostnames.pdf

03.eecs.northwestern.edu or *batman.eecs.northwestern.edu*). Ensure SSH is selected, then press Open. You should get some sort of message asking whether or not you trust the host. Press yes. From here, login as your EECS username (probably the same as NetID), and your EECS password (not necessarily your NetID password). You should now be logged into one of the Northwestern EECS boxes!

(Note that you can configure PuTTY so that you don't have to do all of this every time.)

Mac/Linux

For those of you on Mac or Linux, everything you need is already installed. Open up your terminal and at the prompt type a single command of the form

Mac users: search for "terminal" in Spotlight

```
$ ssh [eecs-id]@[eecs-host].eecs.northwestern.edu
```

Don't type the `$`.

where *[eecs-id]* is your EECS username (probably your NetID) and *[eecs-host]* is replaced by one of the EECS hostnames from the list of student lab hostnames (such as *tlab-03.eecs.northwestern.edu* or *batman.eecs.northwestern.edu*).

http://www.mccormick.northwestern.edu/eecs/documents/current-students/student-lab-hostnames.pdf

You should get a message saying that the authenticity of the host can't be established, and you should be asked if you want to continue connecting. Type "yes" as prompted and press Enter. Now type in your EECS account password (not necessarily your netID password), press Enter again, and you should be logged in remotely!

Basic shell navigation

There are a few basic commands we will be using frequently throughout this exercise in our shell: *cd*, *ls*, and *pwd*, and *man*.

cd stands for "change directory," and is used to change the current directory we are looking at in our shell (our working directory). You can think of a directory as a folder from your regular interactions with your computer. For example the command `$ cd Documents` will look for a directory inside our current directory called Documents, and if it exists, our working directory will become that Documents directory. If you ever want to go back to your home directory, the command `$ cd` with no argument will switch your working directory back to your home directory. The command `$ cd ..` will switch your working directory up one level from where you currently are.

As usual, don't type the `$`.

pwd stands for "print working directory," and is used to print out the current working directory of your shell. For example, if you have been navigating around for a while and you are lost you can type in

the command `$ pwd` and you will see your directory printed out into the shell.

ls is short for the word "list," and is used to list the contents and subdirectories within your current working directory. You can type the command `$ ls` into your shell, and you will see all files and directories within your current working directory.

Play around with these three commands for a few minutes in your shell, and see what directories and files already exist on your EECS box!

man is short for "manual," and is used to access the system manuals. For example, you can read the manual pages for *pwd* and *ls* by running the commands `$ man pwd` and `$ man ls`. To learn about man, you can of course run `$ man man`.

Hit `q` to quit.

Once you are done playing around, type `$ cd` in to navigate back to your home directory. We will be making a new directory for this lab using the *mkdir* command.

Creating new content

mkdir stands for make directory, and is used to create a new directory within our current working directory. For example, `$ mkdir fun-project` will create a new directory inside our current one called fun-project that we can *cd* into if we so desire. We can create hierarchies of directories to keep our files well organized.

Create a new directory inside your home directory called lab1-dir. Change your current working directory to lab1-dir, and we will now practice editing and compiling some C++ files!

The `$ emacs` command in the shell will open up the Emacs text editor. (On Mac/Linux, you will probably want to use `$ emacs -nw` to avoid starting the X Window System..) Pass in a file that you want to edit (even if it hasn't been created yet), and you can start editing that file! For example type `$ emacs -nw my_code.cpp` and you can start editing a file called my_code.cpp within your current working directory.

Text editor preferences can be a fairly contentious issue among software engineers, and if you already have experience with one of Vim or Emacs, feel free to use whichever you already have experience with instead of Emacs. However, for the purpose of this class, we will be teaching using Emacs. Emacs can also seem scary at first, but after you learn a few simple commands, it will quickly start making sense.

Inside your lab1-dir directory create and open a file using Emacs called animals.txt. Note that the .cpp file extension is what we will be using to indicate C++ files. You will see a text editor pop up that does not look dissimilar to a *Notepad.exe* or *TextEdit.app* editor from your Windows or Mac. However, you will notice that clicking a location using your cursor will not move your cursor to where you click :(

Inside this text editor, type in a list of your 3 favorite animals. Once you have typed in your list, you are going to want to save your file so you can use it later. On Emacs, saving is slightly different than

other programs. Instead of using Command- or Ctrl-s, you are going to use Ctrl-x followed by Ctrl-s. (In Emacs, this is spelled "C-x C-s.") This will save your file to your current working directory. Now, we want to close our Emacs window and get back to our Unix shell. In order to close our Emacs window, we will type C-x C-c (that is, Ctrl-x followed by Ctrl-c).

If you are curious about more Emacs commands, there is a nice basic list here: http://www.cs.cornell.edu/courses/cs312/2003sp/handouts/emacs.htm. You can also run an Emacs tutorial inside Emacs. Press C-h t – that is, Ctrl-h followed by t (no Ctrl).

We can ensure that our file was properly created by using the *cat* command in the shell. *cat* is short for "catenate," and prints out contents of a given file. `$ cat` *[filename]* will print the contents of the file to the shell. If you run `$ cat animals.txt` you should see the file you just created on your shell.

Sending files back home (ET phone home)

Now, if everything worked, we are going to want to bring our list of our favorite animals back to our normal computers so we can access it easier, as it is something that is very essential in our lives. We can do this using a program called *scp*.

scp stands for secure copy protocol, and securely copies files to and from a remote box. For example the command `$ scp` *[filename] [remote-directory]* will securely copy *[filename]* to the remote directory. This also works in the reverse direction. And since we are trying to get our list of our favorite animals back to our computer, we will be using that reverse direction (`$ scp` *[remote-file] [local-directory]*).

Mac/Linux

On Mac, open up a second terminal window (This can be done easily by pressing Cmd-N if you are currently on a terminal window).

Navigate to a local directory where you want to keep your animals.txt file using *cd*, and/or create one using *mkdir*. Now, use the *scp* command to bring the file home! This will look something like the following:

```
$ scp [eecs-id]@[eecs-host]:lab1-dir/animals.txt .
```

Note the second argument is a single period, which signifies the current working directory to be the destination address, so *scp* will copy the animals.txt from the remote machine to the current working directory on your local, home machine. For example, my *scp* command looked something like this:

```
$ scp wsc147@batman.eecs.northwestern.edu:lab1-dir/animals.txt .
```

Look around on your local directory to make sure animals.txt exists; if it does you are all done! If you aren't make sure you did things correctly, and if you need help, raise your hand for a peer mentor!

Windows

Using the search menu, search for the command prompt on your Windows machine. Once you have this open, you can navigate to a directory you want to copy your animals.txt to using the same *cd* command as before, but instead of using *ls*, on Windows you can use *dir* instead. Once you have navigated to a directory you are comfortable with (such as the Desktop or Documents or what have you), we will be using the *pscp* command that came with our PuTTY install. This command will look something like this:

```
$ pscp [eecs-id]@[eecs-host]:lab1-dir/animals.txt .
```

Note the second argument is a single period, which signifies the current working directory to be the destination address, so *scp* will copy the animals.txt from the remote machine to the current working directory on your local, home machine. For example, my *scp* command looked something like this:

```
$ pscp wsc147@batman.eecs.northwestern.edu:lab-dir/animals.txt .
```

Look around on your local directory to make sure animals.txt exists, if it does you are all done! If you aren't make sure you did things correctly, and if you need help, raise your hand for a peer mentor!

Creating our build system

Back on our remote logins, please navigate to your home directories (using just the `$ cd` command). Once your current working directory is your home directory, we are going to set up a configuration file so that you will be using the correct version of CMake, and that it will be correct each time you remotely login to your EECS account. Don't worry about understanding what is going on right here, it is something that just needs to be configured for this quarter. To do this we will be creating a file called .tcshrc in your home directory (note the period, and ensure this name is spelled exactly correct). Once this is open, type the following line exactly into your file, save, and exit it:

Remember to create this file and edit it using Emacs we will type in `$ emacs .tcshrc`.

```
source /home/jesse/pub/etc/csh_profile
```

Now, for this time only, type in `$ exec tcsh` into the shell to reload.

Each time you open up your remote connection (including right now) and plan on using CMake (probably every time for first few weeks of the class), type the `$ dev` command into your shell. This will ensure that everyone is on the same (and correct) developer toolset.

Using our build system

As briefly mentioned in class, *CMake* is our build system we will be using for the first few weeks of the course at least. We will usually be giving some sort of starting structure for the projects you will work on, and right now is no exception. This structure is found in a ZIP file which will need to be downloaded onto your remote shells. We have a command called wget which will help download things from the internet for us.

wget stands for "web get," and downloads things from the web using the following command format:

```
$ wget [url]
```

For this example, we will be getting our file from this url: http://users.eecs.northwestern.edu/~jesse/course/eecs211/lab/eecs211-lab01.zip. So therefore, our command will be:

```
$ wget http://users.eecs.northwestern.edu/~jesse/course/eecs211/lab/eecs211-lab01.zip
```

Once we have our ZIP file, we will need to turn it from a compressed file into a new directory, using the *unzip* command. Don't worry too much about what this is, just understand what it does. Use the following command to turn this file into a directory called eecs211-lab01:

```
$ unzip eecs211-lab01.zip
```

Now, once we have our new directory with its files, change your directory to eecs211-lab01 using the *cd* command. Now list its contents using *ls*, and notice that there is a CMakeLists.txt file, and a hello.cpp file. The CMakeLists.txt file is a CMake configuration file which you won't have to worry about too much, and the hello.cpp file is a C++ file which we have provided you.

When using CMake, you will want to create a subdirectory to do your building and create your executable files the first time you set up each project. To do this create a subdirectory called build using *mkdir*, and change that to be your current working directory. Then, the first time you set up each project, you will use the *cmake* command like so: `$ cmake ..` to set up the build environment. This should spit out some commands into the terminal, and if everything is successful, your CMake project is set up!

Note the two periods, signifying the directory up one level, or in this case, the eecs211-lab01 directory.

Now, you can build your program using the command *make*.

The basic purpose of Make is to build your project into an executable file. In your build directory, each time you update your code, you can run

```
$ make [target-name]
```

to create your executable called *[target-name]*. In this case, run `$ make hello` to build a program called hello. In your build directory, if you list its contents, you will see a few more CMake files, but you should also now see a file called hello, which is your executable you can now run:

```
$ ./hello
```

This should spit out a nice greeting.

Updating our code

So, we gave you a basic function and you were able to run it, but how do you change the code?

Navigate back up to your eecs211-lab01 directory using cd (`$ cd ..`). Now open up the hello.cpp file using Emacs, and edit it so it now says "Aloha 211 student!" instead of "Hello 211 student!." Make sure to save and exit Emacs.

Remember C-x C-s to save and C-x C-c to exit.

Go back to our build directory using cd. Try running `$ ./hello` again. Did anything change?

The reason why you still see "Hello 211 student!" on your screen is because while you changed your C++ code, your computer doesn't understand the C++ code, but only the machine code you create by using *make*. So now, run `$ make hello` once again, and try `$ ./hello`. Notice how you now have the correct output! Each time we want to change our code, we are going to need to remember to rebuild our executable. Don't worry if you have error messages your first few times trying to write new code, this is completely normal. Even the best developers in the world usually need a few tries before they can properly build their files, so just take a deep breath, and try and figure out what went wrong.

Conclusion

Knowing how to use the shell is an extremely important tool in computer science. Don't worry if it is still hard for you to use, like much of life, it is one of those things you'll just need to practice with until it seems much more familiar! On your own time, it would be a good idea to continue learning more about the shell and playing around with some more commands. Of course, come to office hours or post on Piazza with any questions or if you want any more challenges!

A good resource for some basic commands is here: http://www.computerworld.com/article/2598082/linux/linux-linux-command-line-cheat-sheet.html.

Useful links

Lab 1 Project http://users.eecs.northwestern.edu/~jesse/course/eecs211/lab/eecs211-lab01.zip

EECS Lab names http://www.mccormick.northwestern.edu/eecs/documents/current-students/student-lab-hostnames.pdf

Simple command line cheat sheet http://www.computerworld.com/article/2598082/linux/linux-linux-command-line-cheat-sheet.html

PuTTY http://www.putty.org/

Nice Emacs guide http://www.cs.cornell.edu/courses/cs312/2003sp/handouts/emacs.htm

Nice Vim guide http://www.openvim.com/