CSE 3100 Fall 2017 Lab 0

August 31, 2017

1 Preface

Since this is lab 0, we are going to assume you know nothing about *nix systems or working on the CLI. If you're confident that you know your way around a terminal, feel free to skip sections of this and go straight to the full assignment. If you're not confident, don't worry, we'll get you there in no time.

2 SSH

The first step on our journey to C mastery is to get connected to the Virtual Machine where we will be doing all of our work this semester. If you are operating on a *nix system (such as MacOS or any flavour of Linux), this is as simple as opening up a terminal and issuing the following command and entering your password (it should be the same password you would use to log onto the computer lab computers)

```
ssh <NETID>@<NETID>.3100.cse.uconn.edu
```

If you are using Windows, you will need to download an ssh client such as putty. The computers in the computer lab should come pre-loaded with it. Once it is installed and launched, you can fill out the host field with "<NETID>.3100.cse.uconn.edu" select the SSH protocol and connect.

3 Woah, where am I?
Navigating The CLI

Once you have successfully authenticated to the Virtual Machine, you'll find yourself looking at a flashing cursor. This is the command line interface of your virtual machine. The most important thing to know about the CLI is that if you're ever not sure how to use a tool, RTM (read the manual). Specifically, read the man pages which you can use by typing 'man' and the name of the tool you want to read about.
Let's go over a few simple commands. If you type ' ls' and then enter, the contents of the current directory (you'll start out in your home directory) will be displayed. Chances are you don't see anything, so let's fix that. Try executing the command

```
touch hello.txt
```

If you execute 'ls' again after that, you should now see that hello.txt file. If you'd like to see what is in a file, you can use the command 'cat' to display the contents. Try that now with

```
cat hello.txt
```

The file is empty, but we can populate it using 'echo'.

```
echo "hello" > hello.txt
```

The echo command tells the CLI to simply print the string we provide it, and the > redirects the output into the file we specify. In this case we print "hello" into hello.txt.
Remember now, if you ever find a command doing something you didn't expect it to do, or want to see what else you can do with a command, read the man pages!

Let's figure out where you are. The 'pwd' command will "Print Working Directory". You should run it now. As you should see, you're still in your home directory. If you want to make a new directory, you can use the command 'mkdir'. Give it a try now, make a directory named "dir1"

```
mkdir dir1
```

If you execute ls again, you should see your new directory. Let's get into that directory now. The command 'cd' changes your directory into the directory you specify. If you don't specify a directory, or ask it to navigate to ~ it will return you to your home directory.

```
cd dir1
```

If you exeute 'ls' now you should see nothing again. It's worth noting that ls can take a path to a directory as an argument, and then it will list the contents of that directory as well. If you want to specify the directory above the one you're currently in, you can use .. as a subsitute for the full path. List the contents of the directory above yours now to make sure that all your files didn't dissapear once you stopped looking at them.

```
ls ..
```

Nice, they're still there. Let's make a copy of hello.txt into this current directory. Do this with the 'cp' command. cp takes a path to a file as the first argument, and the path you'd like to make a copy to as the second argument. To specify the current directory, you can just say . . Do that now

```
cp ../hello.txt .
```

If you ls again, you should see that your new directory has a copy of hello.txt in it. Let's rename it. You can use the 'mv' command to rename/name a file. It works very similarly to cp, let's rename it to hello2.txt.

```
mv hello.txt hello2.txt
```

We're almost done now, there's just one more tool to you'll need. We've made a real mess learning and now we should clean it up. Use the 'rm' command to remove a file. Let's change directory to your home directory first.

```
cd
```

Next, let's remove the files we created in this directory. If you've forgotten the names of the files, use ls. Alternatively, tab will autocomplete words/show you a listing of commands/files that match what you've typed so far. Start with hello.txt.

```
rm hello.txt
```

Now let's try to get rid of dir1

```
rm dir1
```

Oops, you should get an error now. You can't remove a directory without specifying the '-r' option.

```
rm -r dir1
```

Be careful with rm, there is no way to recover a file if you remove it. If you're really careful, you can use 'rm -i' to tell the program to ask for a confirmation.

4 The Wonderful World of Git, Vim (and others...)

Git is the tool we'll be using to submit work in this class, and if you plan on doing any large projects in the future, git will be a very important tool to know. We've initialized repositories for all of you, and you can get your own copy of it using the command 'git clone'.

```
git clone sysprog@ash.engr.uconn.edu:cse3100fa17.<NETID>
```

Now you should see your repository as a new directory in your home folder. If you'd like to rename it or move it, you can do so using the mv command. Enter the lab0 directory within your repository and look around. Inside you should see a Makefile and a hello.c file. We're going to edit the hello.c file. To do this, we need to use a text editor. On the command line, the most common options are nano, emacs, or **vim**. Nano is the simplest, while emacs and **vim** are more powerful. There are many resources available online if you would like to learn how to use them, and of course you should always check the man pages for help. Personally, I suggest you learn **vim**, but you will be okay using nano or even emacs. Regardless of what you choose, open up the file using one of the following commands:

```
emacs hello.c
vim hello.c
nano hello.c
```

The important things to know about each one are listed below:

In **vim**, enter insert mode by pressing i and move using the arrow keys, or, preferably, hjkl. In insert mode you can edit the file. To save the file, press esc to exit insert mode and then type ':w'. To quit the file, make sure you're not in insert mode and thent type ':q'. You can combine the two to save and quit using ':wq'.
In emacs, exit immediately using Ctrl+X and then Ctrl+C, then use **vim**.
That was a joke, I hope you liked it. That being said, in emacs you can start editing immediatly, and when you're done you can use Ctrl+C and then Ctrl+X to attempt to exit. You will be asked if you would like to save, and inputing 'y' will save. You can also use Ctrl+X and then Ctrl+S to save.

With nano, the useful commands are displayed at the bottom of the screen. You can edit immediatly, and save and exit using Ctrl+X and then selecting yes.

Now that that buisness is settled, let's edit your first C file. Using vim or an inferior editor, open hello.c and edit it so that instead of printing "Hello World" it instead prints "Hello C!"
Once you've made the edits, you can compile it using the following command

```
cc hello.c -o hello
```

This tells our compiler to compile the C program hello.c into a binary file named hello. We can execute it by running the command

```
./hello
```

The leading ./ is necessary because the CLI doesn't know where to look for the hello program otherwise, so the ./ tells it to look in the current directory. Once you've verified that you're printing "Hello C!", you should push your changes to the remote repository. This makes it so anyone with access to your repository can see your changes. The general process for pushing changes goes as follows.

```
git status
git add <files>
git commit -m "<commit message">
git push
git pull
```

Git status will show you what files have been modified since you last commited your changes. Adding the files adds files to the list of file that are waiting to be commited. When you commit, you should commit

with a message explaining what has changed since the previous commit. If you're ever lost, you can use the command 'git log' to review commits and commits messages, so make sure to commit descriptively and commit often. When you push, all the commits since your last push get pushed to the remote repository so that they are stored somewhere other than your local machine. Finally, the git pull will pull down any changes that have been pushed to the remote repostiory that are not currently on your local machine. You should run a pull before each lab or assignment so that you get any new files that have been added for that assignment.
That's everything you'll need to know from now until the final... Okay not really, but these are the basic things that you'll need to do for every single assignment, so make sure you know them well.

5 The Final Test

If you skipped here from the start, make sure that you've cloned the repo and edited hello.c so that it prints "Hello C!", and pushed those changes. Now, you should be able to complete the following tasks:

- In the lab0 directory, create two directories, A0 and A1.
- Within A1, create two directories, B1 and B2
- Within B2, create a directory called C1
- Git add all of these new directories
- Commit them with the message "Lab 0 Submission"
- Push them to the remote repository.

To make sure that the pushes have gone through correctly (assuming there were no error messages you didn't ignore, they should have), you can navigate back to your home directory and clone a new copy of your repository. In general there is no need to do this, but if you're ever unsure, do this first and check to see if the directory looks like you'd expect it to before worrying too much.

If you've followed along, you should now be done with the assignment. Call over your instructor and have them check to make sure you've done everything right, and good luck on future labs.