Assignment 1: Setting up & Warming up

CS 3650 Computer Systems, Spring 2022

Due: Thursday, January 27, 10pm

Submission: Submit the contents of your repository via Gradescope. See Deliverables below for what to submit.

This is an *individual* assignment.

Please see Assignment 1 on Canvas for the Github Classroom link.

The purpose of our first assignment is to make sure everybody is able to set up a working environment and to familiarize yourself with the terminal, shell, and the C compiler.

Part 0 - Obtain the repo from Github Classroom

1. Make sure you have use the Assignment 1 invitation link and accept the invitation. This will create a private repository for you called `a1-yourname`, which you can clone and work with offline. We will use this approach for most labs and assignments.
2. Run through the following github tutorial if you are not familiar with git.
   - https://guides.github.com/introduction/git-handbook/
   - This is worth the 15 minutes it takes to avoid headaches later on!
   - Video guides if needed: https://www.youtube.com/githubguides

Part 1 - SSH

In this class, the best is to work using a local x86_64 bit Linux virtual machine. However, being able to use SSH and work in a remote command line environment is part of the basics in this course. Moreover, sometimes it might be easier for you to just use SSH for testing things out.

First you will need to make sure you have an SSH client available on your computer.

ssh clients for each operating system

- For Windows Users, installing PuTTY is sufficient. https://www.putty.org/
  - One option is to use PowerShell. It includes a ssh client.

  - Another option is to use/install Windows Subsystem for Linux (WSL), which comes with a Bash shell and an SSH client. https://docs.microsoft.com/en-us/windows/wsl/install-win10

    The advantage of this solution is that it will give you a Linux development environment directly in Windows. Note, however, that not all what we'll do in this class is completable in this environment and you should still test your work on our VM before submitting.

- For Linux/Mac users, you already have a terminal and an SSH client available! Just search for the 'terminal' application.
- We additionally will install a virtual machine which you may work from, and which has a terminal from which you can ssh from.

ssh'ing

Once your terminal is open, 'ssh' into Khoury with:

```
$ ssh khoury_user_name@login.khoury.northeastern.edu
```

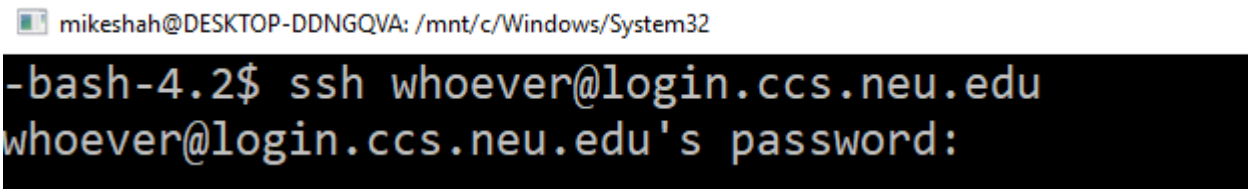
where "khoury_user_name" is the username of your Khoury account, e.g. if "xyzzy" is your Khoury account username, then

```
$ ssh xyzzy@login.khoury.northeastern.edu
```

If, for some reason, you do not have a Khoury username, follow these instructions.

After you have successfully ssh'd, you are now running programs on the Khoury servers (i.e. not locally on your machine).

If you are using PuTTY, follow the instructions for entering the server name (`login.khoury.northeastern.edu`) and your username.



Part 1.5 - Obtaining your repo (i.e. cloning your repository)

When you have successfully ssh'd into your remote account (Part 1) you should download a copy of your repository on the Khoury servers. Before you'll be able to do that, you need to generate an SSH key for yourself. Once you create an SSH key, you can add it both to your Khoury Login account and, later, to your VM instance. The full process is described on Github https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent, here we give the main steps.

Generating a new SSH key

When logged in via SSH to Khoury Login, run the following command:

```
$ ssh-keygen -t ed25519 -C "your_myneu_name@northeastern.edu"
```

You will be asked which file you want to save the key to. The default should be fine, so just press Enter.

Next, you will be asked to pick a passphrase. Choose something secure, but make sure you remember it for later. Confirm the passphrase you picked.

Your whole interaction might look something like this:

```
-bash-4.2$ ssh-keygen -t ed25519 -C "your.username@northeastern.edu"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/fvesely/.ssh/id_ed25519):
Created directory '/home/fvesely/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/fvesely/.ssh/id_ed25519.
Your public key has been saved in /home/fvesely/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:I3ibRo4M+9uLh52BAk/6ftoPWxggMRJ/afd3f7fLLxI your.username@northeastern.edu
The key's randomart image is:
+--[ED25519 256]--+
|=.               |
|.+   .           |
|. o + .          |
|...+ o .         |
| =. o.+ S . .    |
|. o+.B.+ o .E.   |
| ...=o*o     .. o|
|  .ooB+     . o.o|
| .oo*++.     . =+|
+----[SHA256]-----+
-bash-4.2$
```

This step has generated a public & private key pair in the `.ssh` directory. You can check that this is the case by using the `ls` command (more below):

```
-bash-4.2$ ls ~/.ssh
id_ed25519  id_ed25519.pub
```

Finally, run the following commands:

```
-bash-4.2$ echo 'eval "$(ssh-agent -s)"' >> ~/.bashrc
-bash-4.2$ source ~/.bashrc
-bash-4.2$ ssh-add
```

In the last step, enter your chosen passphrase. These last 3 commands will ensure that your ssh key is available when it's needed.

Note: Next time you log in, you only need to run `ssh-add` again.

Adding your SSH key to Github

This part is covered in https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account.

You will need to figure out how to copy text from your terminal or PuTTY. Selecting with the mouse pointer and using Ctrl-C works most of the time. Print the contents of the file `~/.ssh/id_ed25519.pub`, e.g.,:

```
-bash-4.2$ cat ~/.ssh/id_ed25519.pub
```

Select and copy the contents of the file. Log in to your Github account, click on your profile photo in the upper right corner of the page, and go to *Settings*.

Click on *SSH and GPG keys*, then *New SSH key*.

In the *Title* field, give your new key a name, e.g. "Khoury Login Key".

Paste the text from the `id_ed25519.pub` file into the *Key* field.

Click *Add SSH key*. You might need to enter your GitHub password.

The final step is authorizing your new key for use with our Khoury-CS3650 organization. While on the *SSH and GPG keys* page, select *Enable SSO* next to your new key and click the *Authorize* button next to our organization's name. You might need to enter your password again. But that should be it. Now you're ready to clone your repo.

Cloning your repo

Go back to your terminal with ssh running.

1. Run `git clone your_repository_spec` to establish a git repository on your system. *your_repository_spec* is found by clicking the 'green' button on your Lab 1 repository page and selecting 'SSH'.

   It should look something like `git@github.com:Khoury-CS3650/a1-yourname.git`.

   This step will create a local *working directory* with the contents of the repository.

2. When you make a change to a file within this directory you can see what is different by running `git status` to see what you have changed locally on your computer. Changes you have made locally have not yet been saved to Github's servers. You can also see the changes in more detail by running `git diff`.
3. When you are happy with your changes do `git add whatever_file.c` which prepares that specific file to be added to the master. See the git add documentation.
4. Next, you will do `git commit -m "some informative message about your changes"` to record changes in your repository See the git commit documentation. For the future, you might want to learn how to write *good* commit messages.
5. Finally, do a `git push` to actually make things happen–meaning everything you have added will go to the Github server. You can check your github.com repository to see exactly what I will see. See the git push documentation.

Git Cheat Sheet!

This Git Cheat Sheet might be helpful when learning Git.

Part 2 - The (at least) 10 Commands

Try running the following commands (See deliverables section at the end for copying and pasting to output.txt).

1. ls - lists the files and folders in the current directory.
2. pwd - Echos (i.e. prints) the current directory you are in to the terminal
3. mkdir - Create a new directory
4. rmdir - Removes an empty directory
5. cd - Change directory
6. mv - Allows you to move a file elsewhere (sometimes I use this to rename a file)
7. cp - Copies a file
8. touch - Typically I use to create a new empty file that does not exist.
9. man - Manual pages
10. ps - Shows which processes are running
11. echo - Prints out a line of text.
12. whoami - Prints which user you are logged in as.
13. sort - Sorts information
14. cat - Concatenates files and prints them to standard output
15. nl - Outputs a file with number of lines
16. cut - Remove sections from each file
17. grep - Prints lines that match a pattern. This is a very powerful serach command.
18. Other interesting commands/programs include: cut, sed, awk, locate, clear

Note from instructor: I included urls to the commands above, but it will almost always be faster for you to search the man pages within your terminal (and if you do not have terminal access, you will want to use the web version anyway).

Navigating the terminal quickly (Read and try each)

Here are some other nice things to know with the terminal.

- Pressing the up-arrow and down-arrow keys iterates through your command history (There is also a commmand called *history* you can checkout).
- Pressing *tab* autocompletes if it finds a program, command, or file path.
  - Start typing `mkdi` then hit `tab`
  - `tab` can also auto-complete filenames and filepaths, this can be especially helpful!
- Pressing Ctrl+C sends a signal to the terminal to terminate a program if it gets stuck.
  - Type: `grep .` Then press `Ctrl+C` to terminate.
- Pressing Ctrl+Z sends a signal to the terminal to suspend a program and give you back control.
  - You can play with this command by typing in *sleep 10* (which puts the terminal to sleep) for 10 seconds, and see how you can terminate this program.
- Practice getting help by typing `man ssh` into the terminal. (Press q to quit the manual pages).

A bit of precision

Calling each of these 'commands' (i.e. ls, sort, cat, etc.) is not really correct. Each of these is itself a program (typically implemented in C or some other language).

As an example, here is the source code for 'ls': http://git.savannah.gnu.org/cgit/coreutils.git/tree/src/ls.c Each of these commands are part of the coreutils package in Unix. If you look through the source tree, you will additionally find many other programs (i.e. terminal commands you run in your shell) here: http://git.savannah.gnu.org/cgit/coreutils.git/tree/src/.

Part 3 - Get comfortable with an editor and the C compiler

Compile the program `hello.c` using `gcc` to get an executable named `hello` (Hint: look for the option -o in the gcc manual). Make sure it runs and prints what you expect.

Now open your terminal editor (vim, Emacs, nano, joe, ...) and modify the program so it prints: your first name followed by a newline and your email address followed by a newline.

Recompile the program and make sure it runs.

Here is a little tutorial on VIM as a resource: https://www.howtoforge.com/vim-basics.

Here is a Vim Basics in 8 Minutes video.

Note: You are not required to use the same terminal editor I use (I'm a VIM user), but you should become comfortable using at least one terminal editor in this course and we recommend either VIM or Emacs, both powerful highly customizable editors with huge user bases. In general, it is good to have at least basic knowledge of VIM, since some version of it is installed by default on virtually any Unix/Unix-like system.

Part 4 - Install a VM to work locally.

Now we will install a VM for you to develop and, more importantly, test your submissions in the same environment as we will be grading them. Once you have your VM installed, use the appropriate means to log in and repeat the steps from Part 1.5 to clone your first repository.

Option 1 (for all Intel and AMD devices):

Khoury Systems has released a "Vagrantfile" you can use to set up a virtual machine prectically identical to the environment that will be used to grade your submissions and is similar to the environment running on Khoury linux machines.

Follow the instructions at https://service.northeastern.edu/tech?id=kb_article&sys_id=6fc66992db096fc084ba5595ce961907 to install **VirtualBox** and **Vagrant**. **STOP after 'Verifying the Installation'.**

Do not use the Vagrant file linked there. Instead, follow the directins below to download and initialize our course's Vagrant box. Change to the top level directory where you store the files for this course, then follow the instructions below.

This VM has newer software than Khoury login and runs the Ubuntu version of Linux. Software is installed using `sudo apt-get install <package name>`.

```
# This downloads the box image
$ vagrant box add khoury-cs3650/base-environment

# This will create Vagrantfile in your working directory
$ vagrant init khoury-cs3650/base-environment

# Starts the box defined in the Vagrantfile
$ vagrant up

# This will ssh directly into the vagrant box (Press CTRL+D to exit)
$ vagrant ssh
```

Note: After booting up the VM, always `cd /vagrant` and do your work in there. This will ensure that all files you create in the VM will also be available on your host filesystem.

Troubleshooting on Windows On Windows, you might run into an error when running `vagrant up`, that looks something like this:

```
Command: ["startvm", "4a213de0-5ac7-4517-93ff-2978b36ad3b2", "--type", "headless"]

Stderr: VBoxManage.exe: error: RawFile#0 failed to create the raw output file
    /dev/null (VERR_PATH_NOT_FOUND)
VBoxManage.exe: error: Details: code E_FAIL (0x80004005), component ConsoleWrap,
    interface IConsole
```

If this is the case, open the `Vagrantfile` in a text editor and add the following lines *to the very end*:

```
config.vm.provider "virtualbox" do |vb|
  vb.customize [ "modifyvm", :id, "--uartmode1", "disconnected" ]
end
```

Try running `vagrant up` again.

Option 2 (for non-Intel machines, e.g. Apple M1 chip MacBooks)

1. Install QEMU. By far, the easiest way to install QEMU under macOS is to first install Homebrew. Then installing QEMU (and many other packages) is as easy as

   ```
   $ brew install qemu
   ```

2. Once you have QEMU installed, create a directory for your VM and copy `systems-qemu.sh` from this repository into that directory. Make sure the `wget` command is installed:

   ```
   $ brew install wget
   ```

3. Enter the directory and run `systems-qemu.sh`. The machine should boot up – after downloading and setting up an image (this might take a while). You can either log in in the VM window (but Copy and Paste will most likely not work for you), or you can allow SSH access (strongly recommended):

   1. Log into the VM using the credentials below.
   2. Open `/etc/ssh/sshd_config`. E.g.

      ```
      $ sudo vim /etc/ssh/sshd_config
      ```

   3. Find the line starting `PasswordAuthentication` and make sure it ends with `yes` (as opposed to `no`).
   4. Re-start the ssh service:

      ```
      $ sudo service ssh restart
      ```

   Now you should be able to log in to your running VM using ssh from you macOS terminal as follows:

   ```
   $ ssh -p 2200 vagrant@localhost
   ```

The credentials for the QEMU VM are:

- username: `vagrant`
- password: `cs3650f21-base!`

More resources to help

- There is a lovely user manual on Virtual Box here: https://www.virtualbox.org/manual/
- There is another example of installing an older Ubuntu on an older version of virtual box here with pictures: https://askubuntu.com/questions/142549/how-to-install-ubuntu-on-virtualbox
- Installing Ubuntu on virtual box in windows video tutorial: https://www.youtube.com/watch?v=IVquJh3DXUA
- If you are not comfortable with compiling a C program on the terminal, take a look at this resource: https://www.cs.fsu.edu/~myers/howto/g++compiling.txt
- Hackerrank has an excellent series of shell tutorials here: https://www.hackerrank.com/domains/shell/bash
  - I highly recommend going through them if you are not comfortable with the shell.
- **If you accidently delete something** you can navigate to `cd .snapshot/` which will show files that have been periodically backed up for that current directory. This is yet anothe reason to make sure you are working within the Khoury systems which provide this backup service.
- A pretty good introduction to shell scripting is here: https://www.howtogeek.com/67469/the-beginners-guide-to-shell-scripting-the-basics/.

Deliverables

1. Copy and Paste the output from the command-line interface of you running five different terminal commands above into a file called `output.txt` (no capitalization, exactly as named.)
   - Make sure to 'add/commit/push' this file to the repo in this directory.
2. Modify the file called `hello.c` in your repository as instructed above.
   - Make sure to 'add/commit/push' this file to the repo in this directory.
   - DO NOT add or commit the executible **hello**.
3. Take a screenshot of your virtual environment running. Name it `linux.jpg` (or `linux.png`)
   - Make sure to 'add/commit/push' this file to the repo in this directory.
4. Create a text file called `repo.txt` containing a single line with the name of your A1 repository repository (e.g., `a1-sp22-ferd`)
   - Make sure to 'add/commit/push' this file to the repo in this directory.

Finally, go to Gradescope and submit your repository (or a ZIP archive which can be downloaded from Github)

Going Further

(Some more terminal programs to research and try out on your own time)

1. history
2. tree
3. scp
    - Copy a file(e.g. backup_copy.txt) from a remote host(i.e. a server) to a local host(i.e. your machine)
        - `scp username@from_host:backup_copy.txt /mike/desktop/`
    - Copy file(e.g. linux.jpg) from local host(i.e. your machine) to a remote host (i.e. a server)
        - `scp linux.jpg username@to_host:/remote/directory/`