

CS330 Lab01: Environment Setup and Hello World in C

Hello and welcome to CS330: Computer Architecture and Assembly Lab. Today we will only be setting up our environment and writing/running a simple C program, to make sure everyone is prepared for the coming semester. We will also be giving a rundown of the rules and structure of the lab.

VPN Setup:

The Vulcan server used to ensure everyone in the class is working in the same development environment was made inaccessible to external users in March 2021. In order to access Vulcan (which will be required for most of the semester), you must connect to UAB's VPN first if you are outside the UAB Secure (Campus) Network. Follow the directions here:

<https://www.uab.edu/it/home/tech-solutions/network/vpn>

You will log into the VPN in a similar way to logging into Blazernet, providing your UAB login information and potentially a secondary password pulled from the Duo application or a text message.

Environment Setup:

A large portion of the work completed in this lab will be programs written and compiled in C or Assembly language. To do this, we will be using a nonlocal Linux environment. Some of you may be familiar with *the Vulcan* server (sometimes called "moat" for its url, moat.cis.uab.edu), but for those of you who are not, we will walk you through connecting to and using the Vulcan server.

Begin by **installing VSCode (outlined in a separate document)**, and the following extensions:

- Remote Development (by Microsoft)
- C/C++ (by Microsoft)
- GNU Assembler Language Support (by Bas du Pre)

Workflow

The following steps will be completed in Lab01, but for reference, the general workflow we will use for all Labs and Homeworks is:

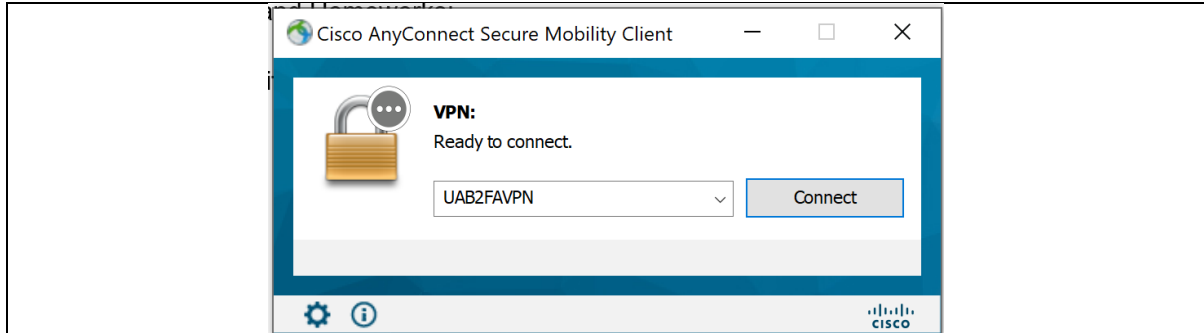
- 1) **(if not on the UAB Secure Network) Establish VPN Connection**
- 2) **Open VSCode, and Connect to Vulcan**
- 3) **Create Folder (if desired):**
- 4) **Create File(s):**
- 5) **Write Code:**

- 6) **Compile:**
- 7) **Debug and Correct any errors, repeat Steps 5-7 as necessary**
- 8) **Download and Submit to Canvas**

Detailed steps below:

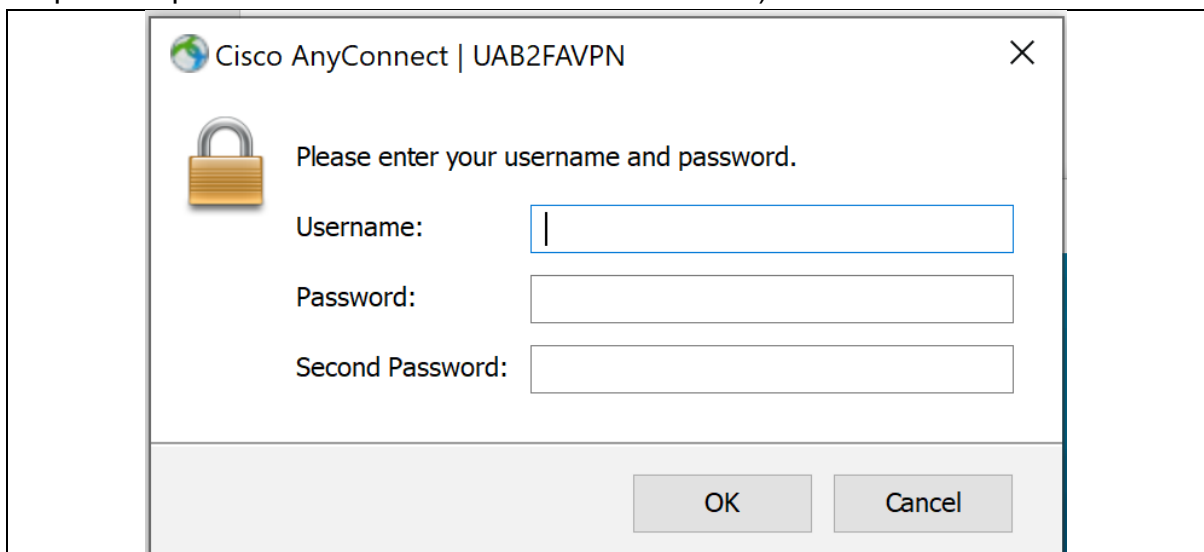
1) (if not on the UAB Secure Network) Establish VPN Connection:

- a) Open CISCO AnyConnect

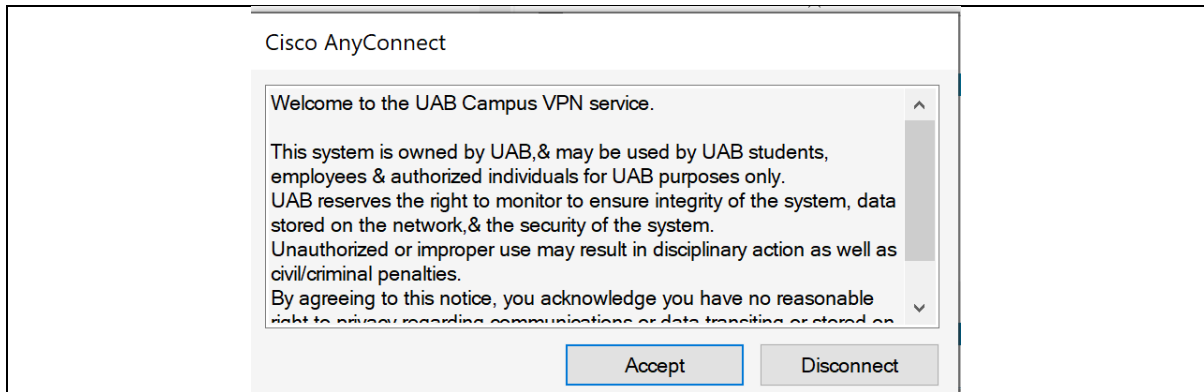


- b) Click 'Connect' button

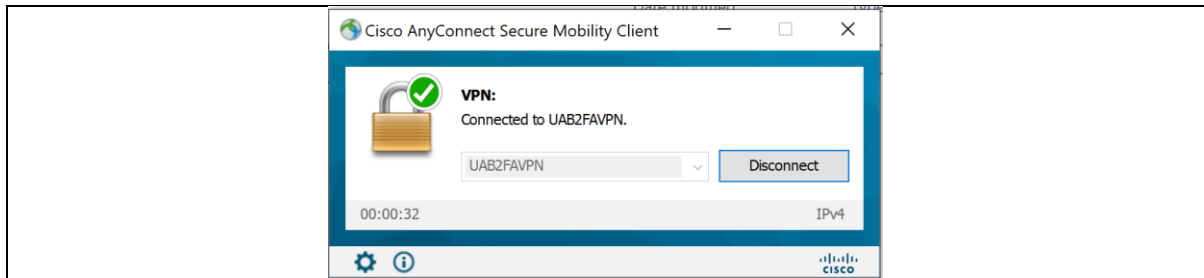
- c) Enter BlazerID, Password, and the word 'push' in the Second Password field (to push request to DUO for two-factor authentication)



If successful, you should see something like this:

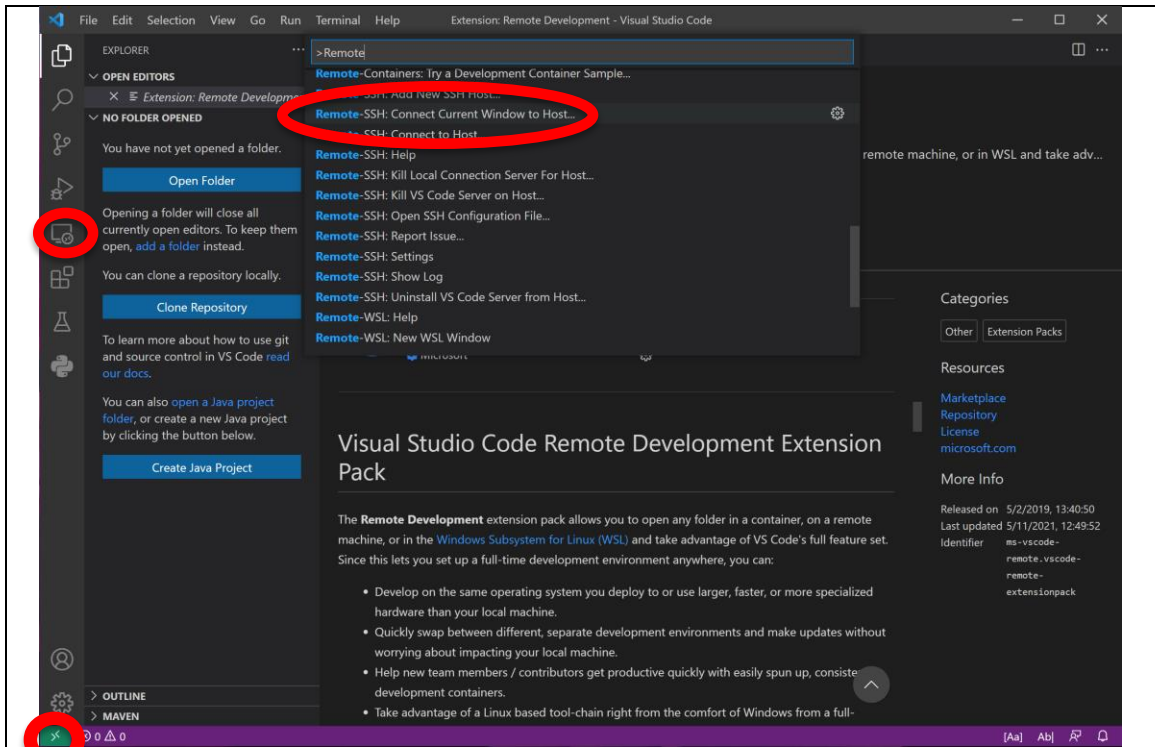


And after clicking 'accept'



2) Open VSCode, and Connect to Vulcan

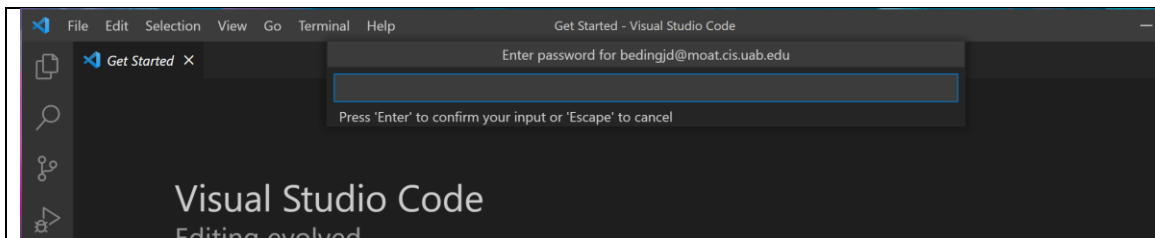
- a) Click on the green connection icon (lower-left),
or Click on the Remote Development icon (left side), right click on moat folder,
and select "Connect ..."
or or type <F1>, "Remote-SSH: Connect Current Window to Host"



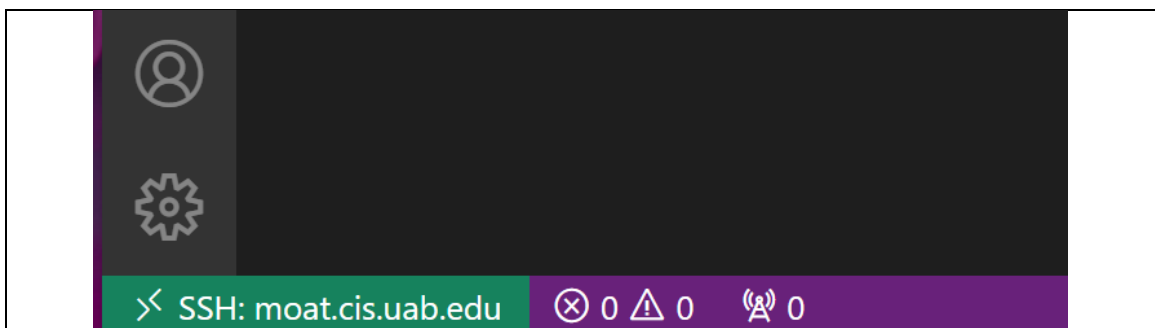
b) (If this is the first time using VSCode Remote Development), set-up the host as moat.cis.uab.edu

c) Select Vulcan (moat.cis.uab.edu)

d) Enter BlazerID Password

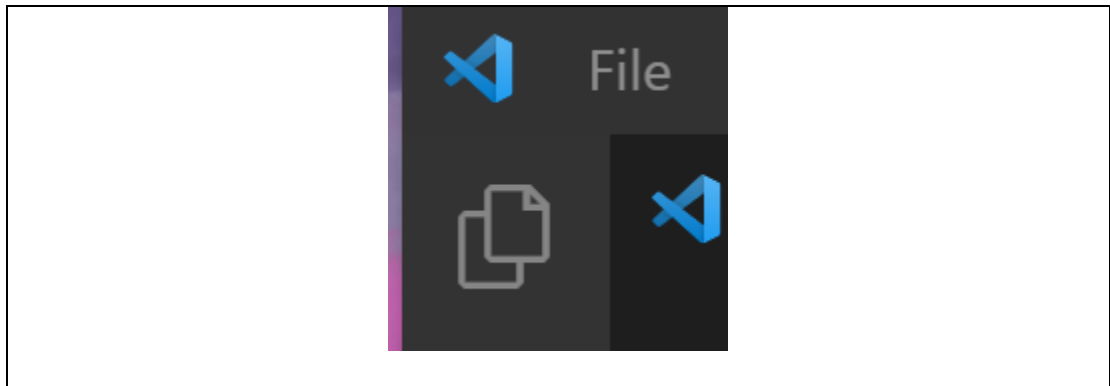


e) When connected, green status is lower left will indicate SSH: moat.cis.uab.edu

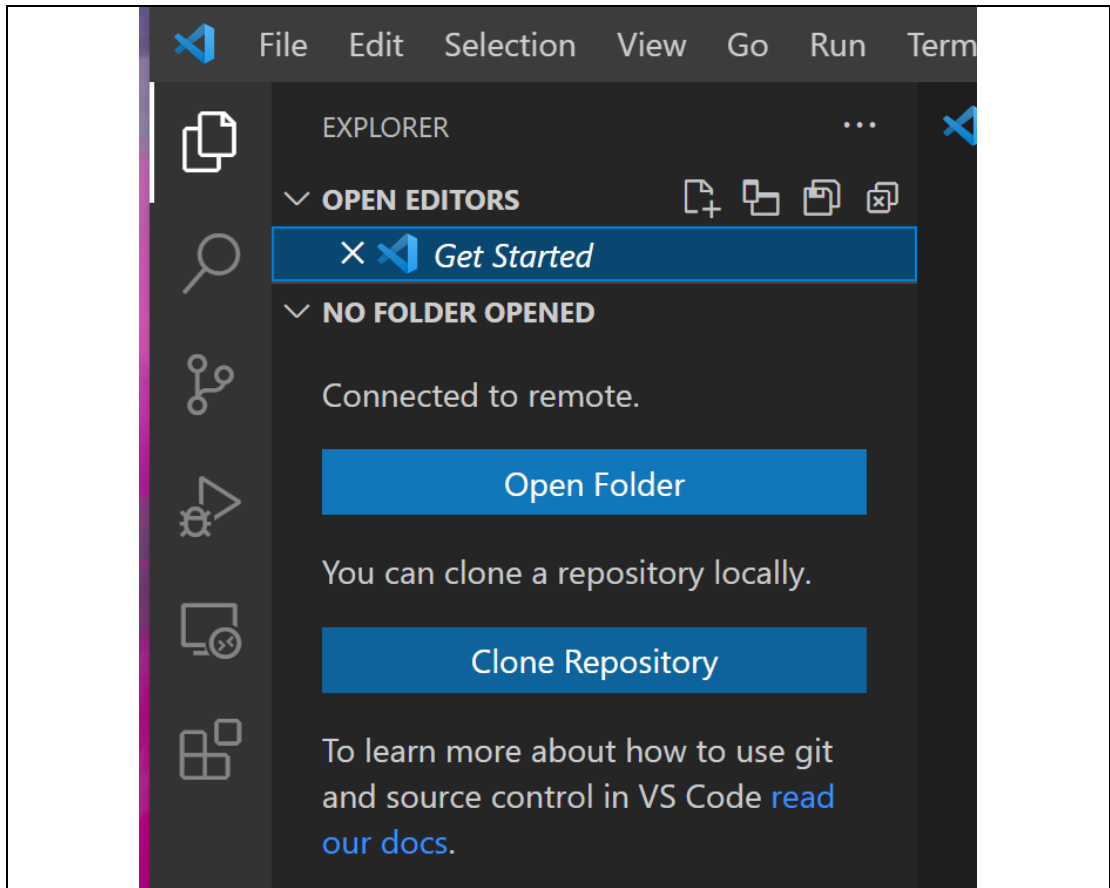


f) Open Folders

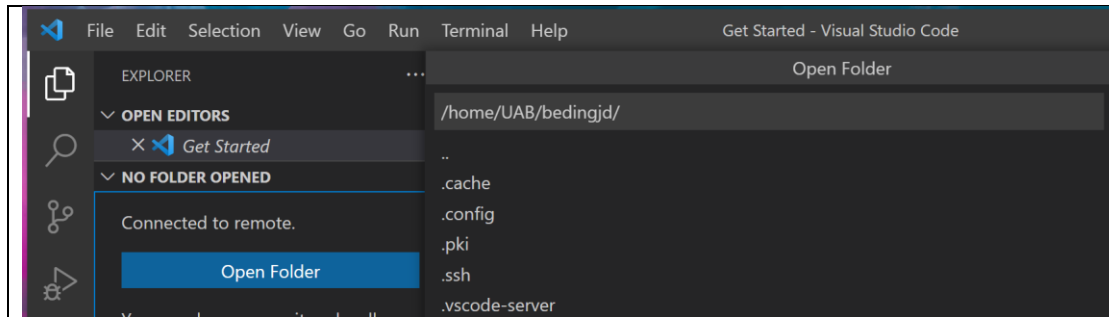
- i) Click Folder icon



- ii) Click "Open Folder"



- iii) Your Home folder should already be the default option, click "ok". If it's not the default option, type "/home/UAB/<blazerid>"

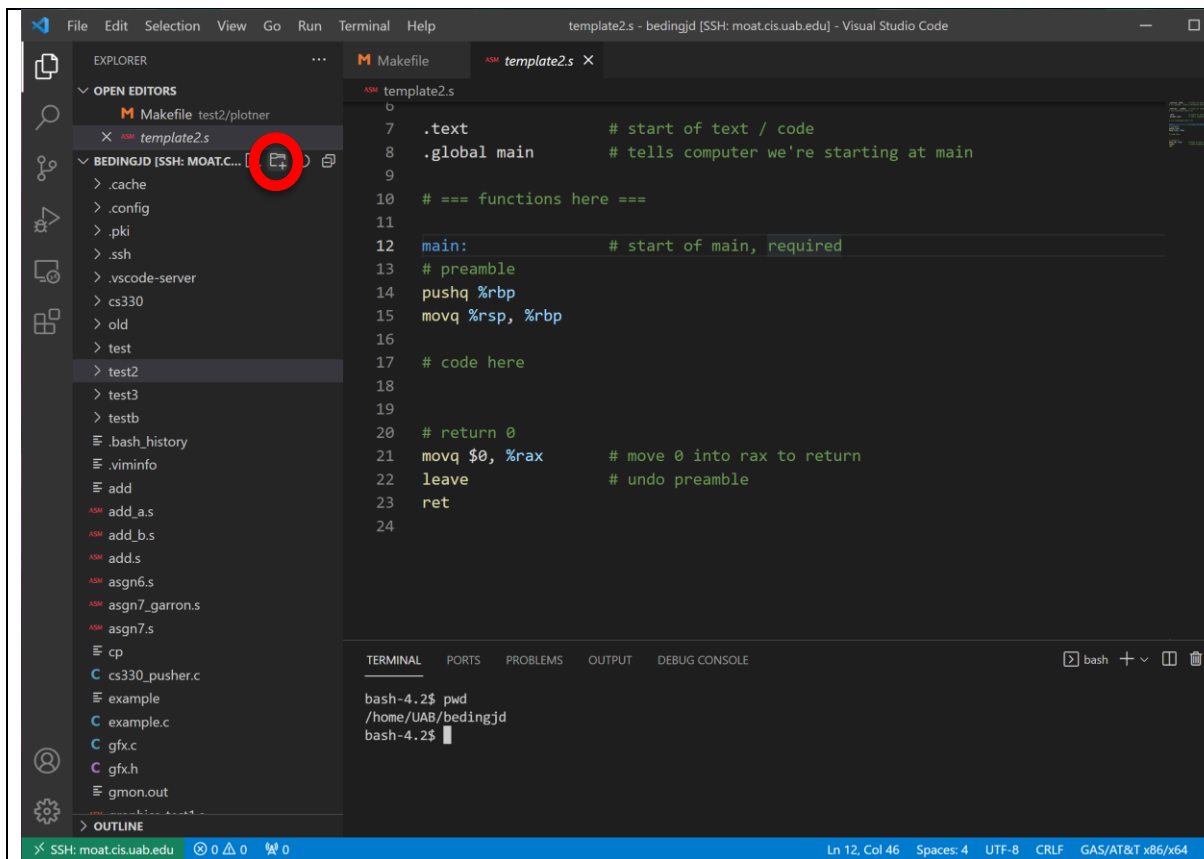


iv) Enter BlazerID password in the prompt

v) Your Vulcan folder(s) should be visible on the left

3) Create Folder:

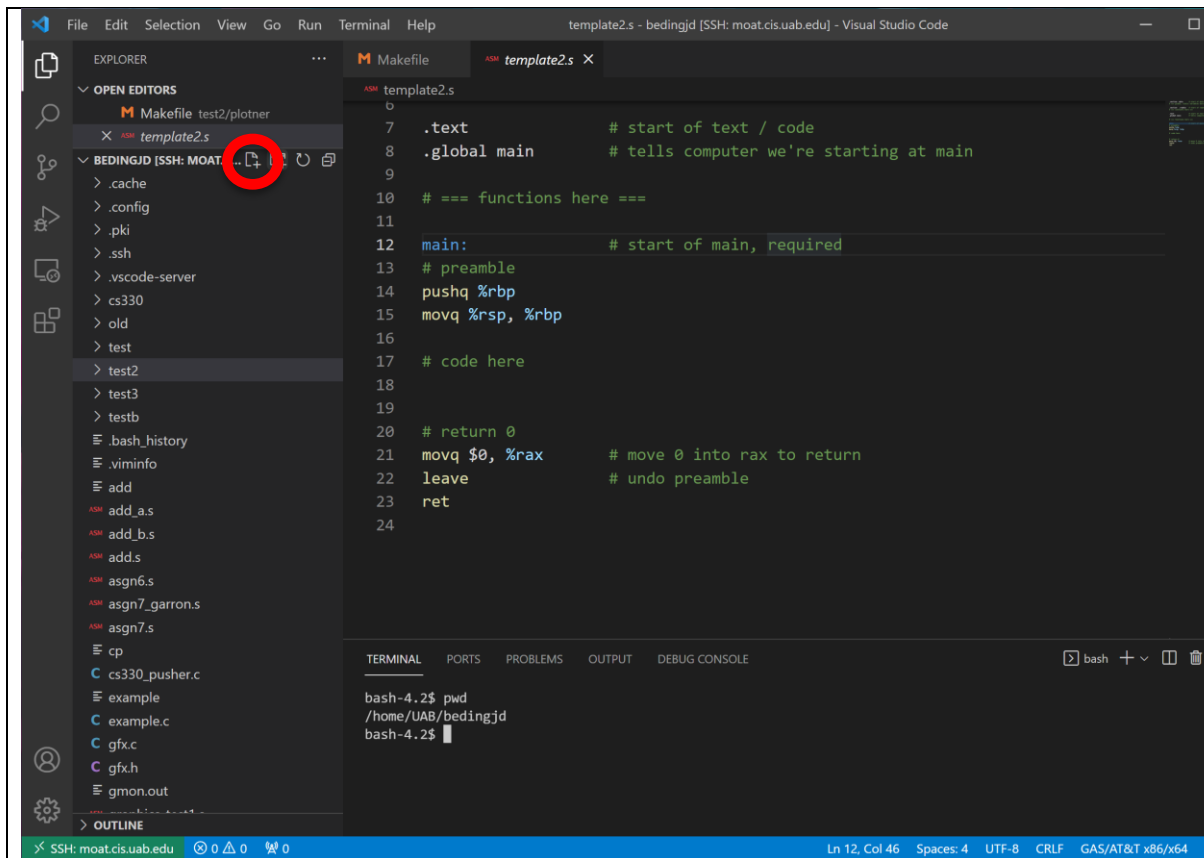
Click on the 'new folder' icon in your folders window:



4) Create File(s):

Click on the 'new file' icon, name appropriately.

For Lab01, name the file hello.c



5) Write Code:

For lab01 we'll write our Hello World program (the original). Write the code as follows:

```

1  #include <stdio.h>
2
3  int main(){
4      printf("Hello World! \n");
5      return 0;
6  }

```

Save the file

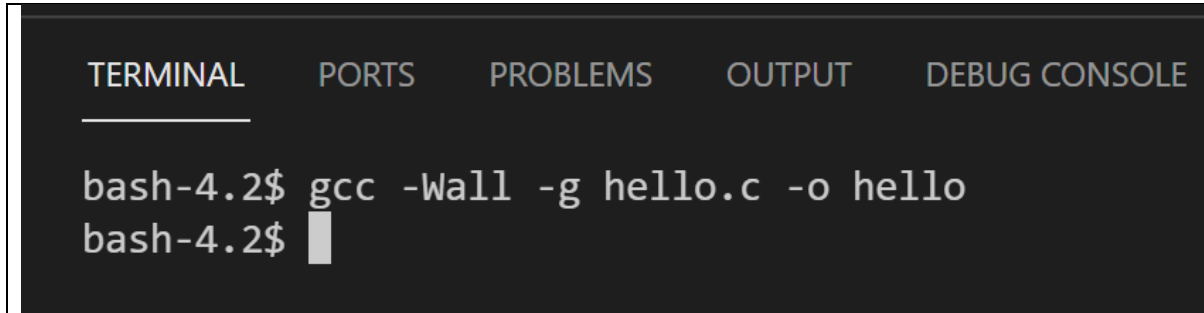
6) Compile:

(in Lab2 we'll introduce Make). For Lab01:

In the Terminal type

`gcc -Wall -g hello.c -o hello`

If there are no compiler errors you should see this

A screenshot of a terminal window with a dark background. At the top, there are five tabs: 'TERMINAL', 'PORTS', 'PROBLEMS', 'OUTPUT', and 'DEBUG CONSOLE'. The 'TERMINAL' tab is selected and underlined. The terminal shows two lines of text: 'bash-4.2\$ gcc -Wall -g hello.c -o hello' followed by 'bash-4.2\$' with a cursor.

To run, type this:

`./hello`

If successful, you should see this, congrats!

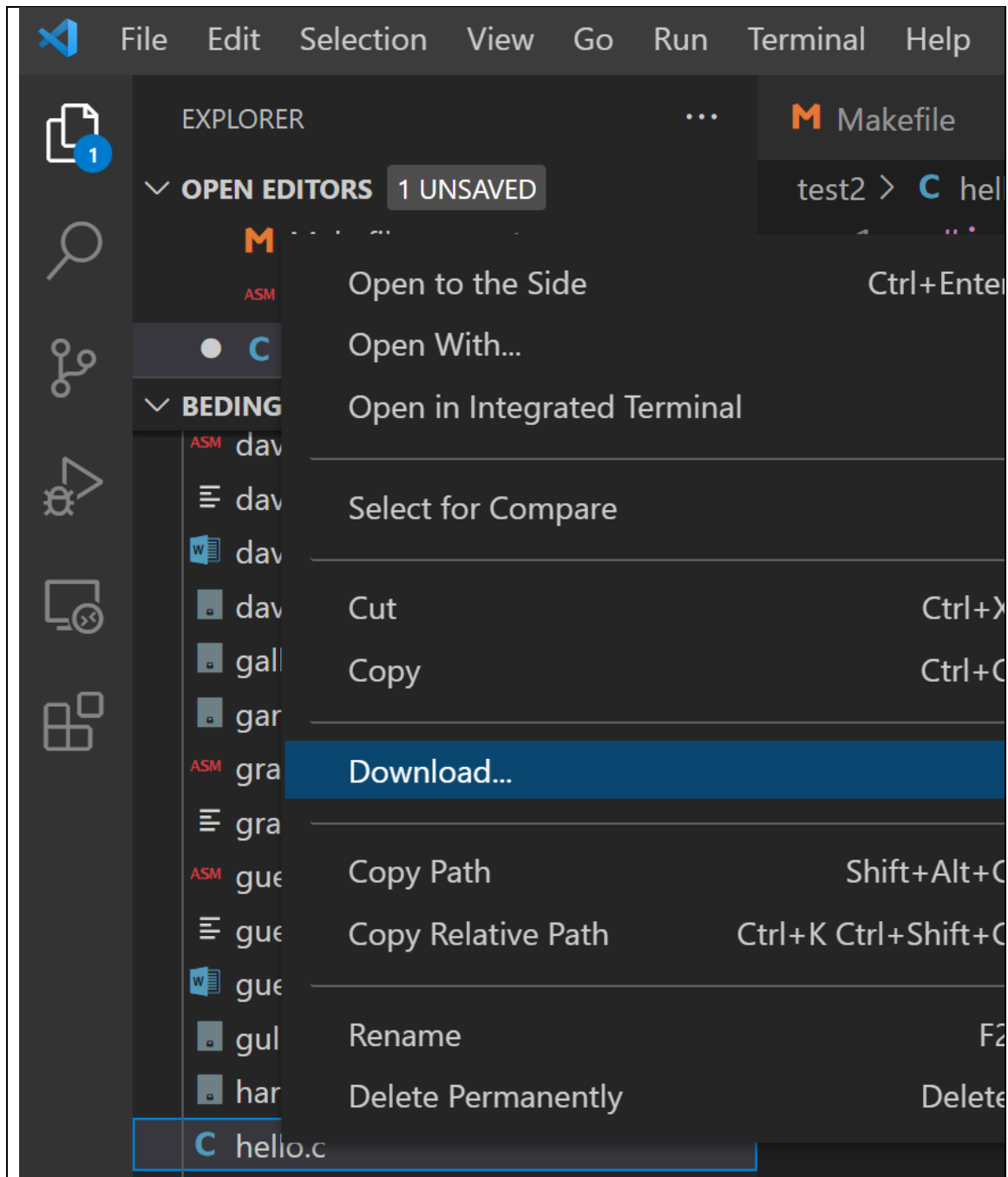
A screenshot of a terminal window with a dark background. At the top, there are five tabs: 'TERMINAL', 'PORTS', 'PROBLEMS', 'OUTPUT', and 'DEBUG CONSOLE'. The 'TERMINAL' tab is selected and underlined. The terminal shows three lines of text: 'bash-4.2\$ gcc -Wall -g hello.c -o hello', 'bash-4.2\$./hello', and 'Hello World!'. The final line is 'bash-4.2\$' with a cursor.

If you don't see this, then proceed to Step 7, otherwise continue to Step 8

7) Debug and Correct any errors, repeat Steps 5-7 as necessary

8) Download and Submit to Canvas:

a) Right-click on your file in the Folders window, select "Download"



b) Chose a location to download

c) Open Canvas, select the appropriate assignment, and upload, don't forget your Independent Completion Form as well.

First Assignment

In this lab, we learned how to connect to a remote device via ssh, send files to and from that device, and how to write and compile basic C programs.

Your first assignment is to compile the Hello World program demonstrated in this lab on Vulcan, and upload the binary executable it creates to Canvas.

Note that this **MUST** be compiled on Vulcan: if you compile on your own system, or even another Linux device, the file will not necessarily be executable on Vulcan, where we will be testing. **If your file is not compiled on Vulcan, it will almost certainly fail to execute on Vulcan, and you will receive a 0 on the assignment if that is the case!**

Grade: Pass/Fail (Pass: HelloWorld **compiled binary** works)

Other requirements: Independent Completion Form, turned in alongside all other assignment components.

METHODS USED IN PREVIOUS SEMESTERS

The following instructions have been provided to Students in previous Semester's CS330. If you've successfully completed the steps above, you don't need to complete every step below. However, even if only use VSCode, be sure you review the material below, and are comfortable working with the command line.

You might also want to try out some of the other editors, and if you're on Windows, highly recommend Windows Subsystem for Linux (WSL).

Accessing Vulcan outside of VSCode Remote Development

If you decide to access Vulcan outside of VSCode Remote Development, you can do so via the following:

First, you will need a Terminal. **Mac and Linux Users** can simply use their installed terminal. **Windows** users can use Windows PowerShell commands, though compatibility is NOT guaranteed. Instead, try using Windows Terminal, Windows Subsystem for Linux, or Cmder, given below. Windows Terminal is the recommended, native solution. Windows Subsystem for Linux is a close second, and is the closest a Windows user can get to a Linux environment. **PLEASE NOTE THAT THE STANDARD WINDOWS COMMAND PROMPT WILL NOT WORK!**

These options are ordered by ease of use, from easiest to hardest. However, WSL (3th option) is still very valuable.

Git Bash: <https://git-scm.com/downloads>

Windows Terminal:

<https://www.microsoft.com/en-us/p/windows-terminal-preview/9n0dx20hk701>

Windows Subsystem for Linux:

<https://docs.microsoft.com/en-us/windows/wsl/install-win10>

Once you have a terminal application to use, you will connect to Vulcan (moat) via the following command:

```
ssh <blazerid>@moat.cis.uab.edu
```

Your login will **be the same as your normal BlazerID and password**. For security reasons, you will be unable to see your password or any characters as you enter it; it is still working! This is the expected behavior on a Linux system.

If you are successful in connecting, you will see something like the following:

```

#####: System Data :#####
+ Hostname      = vulcan18.cis.uab.edu
+ Address       =
+ OS            = Centos 7 amd64
+ Kernel        = 3.10.0-862.3.3.el7.x86_64
+ Uptime        = 21 days
+ CPU           = 2 x Intel(R) Xeon(TM) CPU 3.20GHz
+ Memory        = 3.87 GB
+
+ # of Users    = 1
+
#####: User Data :#####
+ Username      = cobrian
+ Sessions      = 1
+ Processes     = 4
#####: Maintenance Information :#####
Dijkstra probably hates me
(Linus Torvalds, in kernel/sched.c)
Centos Vulcan Test Environment
#####

~~{0.00}~{cobrian@vulcan18:~}~~
$ 

```

Navigating the Linux terminal:

What you're looking at is your home directory on the Vulcan server. This is where you will start each time you access Vulcan.

Some of you may be familiar with a Linux terminal and its commands. If not, here is a list of commonly used commands. I recommend taking some time and getting used to navigating the terminal, and making a cheat sheet of commands while you learn.

- `ls` - list the files in the current directory
- `pwd` - displays the full pathname of the current directory
- `cd name` - change directories into the target directory
 - There are special names for certain folders in Linux, which you need to know!
 - `..` is the parent directory
 - `.` is the current directory

```
- ~ is your personal home directory
- / is the top-most folder in the directory tree i.e. "root" folder.

•mkdir name - create a new directory name within the current directory

•touch name - create a file name (usually used for files with .c or .asm extensions)

•nano name - open a plaintext editor to display or make changes to the project name

•rm name - delete file name in directory (be careful: it will be permanently gone!)
  - Add the argument "-r" to delete recursively, i.e., directories and their contents
•cp file1 filename_or_dir - copy the file file1 to the file file2, or copy it into the
  directory (if file2 already exists, this overwrites the present contents); file1 still
  exists.
•mv file1 filename_or_dir - rename the file file1 to file2, or move it into the directory
  specified (move it to a new file, file1 is no longer in its original location)

•clear - clear the terminal of all content

•man - shows the manual of the input command. For example: man cd

Non-command capabilities:

•pressing tab - autocomplete the currently typed command, if applicable. For example:
mkd<tab> -> mkdir

•arrow keys - up and down to scroll through recently run commands, left and right to
navigate within a given command
```

Once you grasp the above, navigate to your home directory and create a directory *lab1*. Once you have done that, *cd* into the directory. Once that is done, you are ready for today's exercise.

Hello World in C:

Now that we have established our Linux environment and familiarized ourselves with the terminal, it's time to write our first C program: The standard "Hello World" program.

Start by navigating to the Lab1 folder that you created in the last exercise. Use the *touch* command to create a new C file named *hello.c*. If you've done this correctly, you will be able to see the newly created file when you list the files in the directory using the *ls* command.

For example, your terminal should look something like the screenshot below

```

~~{0.00}~{cobrian@vulcan11:~}~~
$ cd lab1
~~{0.00}~{cobrian@vulcan11:~/lab1}~~
$ ls
~~{0.00}~{cobrian@vulcan11:~/lab1}~~
$ touch hello.c
~~{0.00}~{cobrian@vulcan11:~/lab1}~~
$ ls
hello.c
~~{0.00}~{cobrian@vulcan11:~/lab1}~~
$

```

As you can see, the initial list command found no files in the directory we made. After creating the c file with the *touch* command, it can be seen in the directory.

Selecting an Editor:

For the purposes of this lab, we will be using the “nano” editor to change the hello.c file we just created. Nano allows you to edit files within the terminal. It is very useful for small projects and quick changes, but it is recommended that you use an external text editor for later projects and labs. The following are programs you can get for your own system:

- **Notepad++:** <https://notepad-plus-plus.org/downloads/>,
- **Atom:** <https://atom.io/>
- **Visual Studio Code:** <https://code.visualstudio.com/>

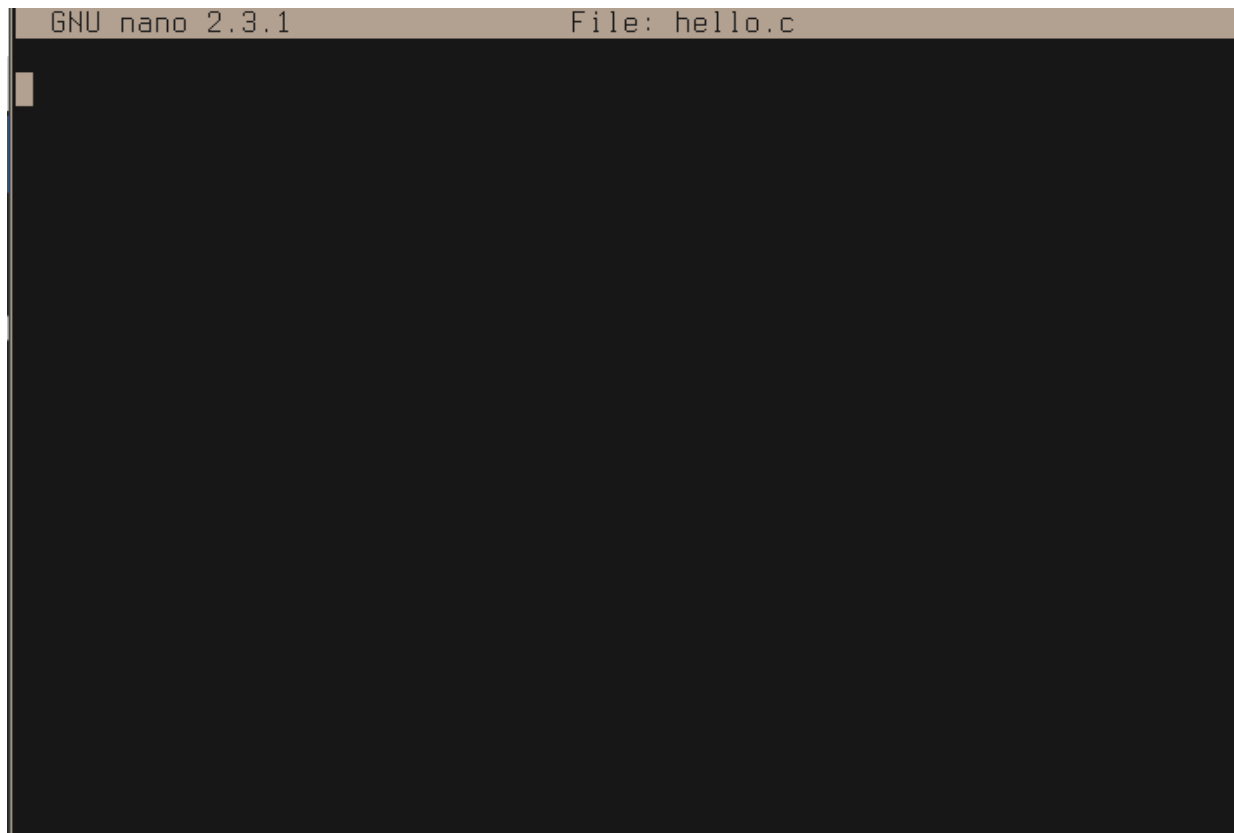
If you are interested in a more advanced and versatile editor, try out the following:

- **Vim:** <https://www.vim.org/> or type “vimtutor” in a terminal such as on Vulcan
- **Emacs:** <https://www.gnu.org/software/emacs/>

Nano is not a powerful or efficient editor, but it is lightweight and on almost every Linux system. Notepad++, Atom, and VScode can all have plugins to make them more capable, but they are largely limited to editing files on your own system which you will have to copy elsewhere. Vim and Emacs are comparably as powerful and configurable as Notepad++, but are available on almost every Linux system like Nano is. Each of these options has their own advantages and disadvantages, so I recommend investigating these options later.

Continuing with the lesson:

Open “hello.c” in your terminal with “nano hello.c”. You should see a screen that looks like this:



Once you see this screen, type the following code into the document:

```
#include <stdio.h>
int main(){
    printf("Hello World!! \n");
    return 0;
}
```

In the nano editor, your code should look something like this

```
GNU nano 2.3.1      File: hello.c

#include <stdio.h>
int main() {
    printf("Hello world! \n");
    return 0;
}
```

Once you have this code written, use CTRL+X to exit nano.

NANO WILL ASK YOU IF YOU WANT TO SAVE. YOU WILL NEED TO HIT “Y” TO SAVE THE CHANGES, THEN PRESS “ENTER” TO CONFIRM THE FILENAME. You will be returned to the terminal once you have done this. You should be able to see the hello program in your directory still. Now you can compile and run it!

Running programs from the terminal:

In order to run C programs from the terminal, you will need to compile them with gcc:

gcc input.c -o output

Where “input.c” is the name of your program, and “output” is the name of the resulting executable. This command will invoke the GNU C Compiler (“gcc”) to compile your program into an output binary (-o). The result of this command is an executable with the specified name; note that you can make this any name, or omit the “-o” argument to get an output file named “a.out” on Unix systems and “a.exe” or “a.bin” on Windows systems.

Second, execute the created file: `./output`

The “./” refers to the current directory, and the rest of the command is the name of the file you specified.

Once you've done this, your program should run and print "Hello World!" to the screen. You can see an example in the screenshot below:

```
~{0.00}~{cobrian@vulcan1:~/lab1}~  
$ gcc -o hello hello.c  
~{0.00}~{cobrian@vulcan1:~/lab1}~  
$ ./hello  
Hello World!!  
~{0.00}~{cobrian@vulcan1:~/lab1}~  
$
```

One more tool you will need

In order to move files back and forth between your personal computer and Vulcan, you will need one more command. It functions very similarly to `cp` which we discussed before. To review, the `cp` command works like the following:

You may copy a given file to an existing folder, where it will have the same name.

cp filename ./other/location

You may also copy the file, changing its name in the process.

cp filename new_filename

You may combine these two:

cp filename ./other/location/new_filename

And finally, you may copy multiple files in one operation to a new location (but you cannot rename).

cp file1 file2 file3 ./other/location

The scp command works the same way as the cp command, except that you can copy files over the internet! And these internet locations are described in nearly the same way as ssh.

**YOU MUST LOG OFF OF VULCAN TO USE SCP
AS DESCRIBED HERE. VULCAN CANNOT SEE
YOUR COMPUTER, ONLY YOU CAN SEE
VULCAN. YOU HAVE BEEN WARNED!**

scp filename username@hostname:directory

There are several arguments: one or more files to copy (if you specify filename more than once), and a destination to copy them to.

The difference here is that the destination is a location over the internet, complete with your username on that server, and the location on that server after a colon.

If we were to copy a file on our computer to Vulcan, it would look something like this:

```
scp hello.c gregdan3@moat.cis.uab.edu:~/lab1/
```

This will copy the file named “hello.c” into the lab1 folder inside my home folder on Vulcan. As you might expect, it would fail if you didn’t have a folder named “lab1” on Vulcan!

In the reverse case, if we copy a file from Vulcan to our computer, it would instead look like this:

```
scp username@hostname:directory filename
```

```
scp gregdan3@moat.cis.uab.edu:~/lab1/a.out .
```

This will copy the file named “a.out” from the lab1 folder in my home folder on Vulcan, into the current directory on my computer. Look carefully, there is a “.” after the command which represents the destination: the current directory!

First Assignment

In this lab, we learned how to connect to a remote device via ssh, send files to and from that device, and how to write and compile basic C programs.

Your first assignment is to compile the Hello World program demonstrated in this lab on Vulcan, and upload the binary executable it creates to Canvas.

Note that this **MUST** be compiled on Vulcan: if you compile on your own system, or even another Linux device, the file will not necessarily be executable on Vulcan, where we will be testing. **If your file is not compiled on Vulcan, it will almost certainly fail to execute on Vulcan, and you will receive a 0 on the assignment if that is the case!**

Grade: Pass/Fail (Pass: HelloWorld **compiled binary** works)

Other requirements: Independent Completion Form, turned in alongside all other assignment components.