

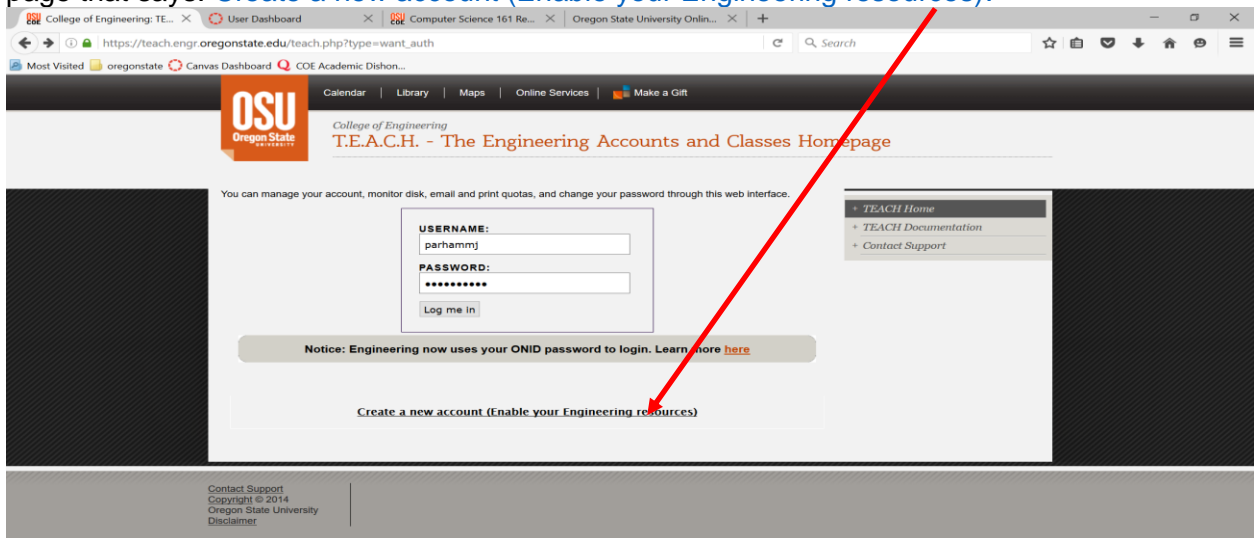
## Lab 1

In order to get credit for the lab, you need to be checked off by the end of lab. For non-zero labs, you can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the lab before the next lab. For extenuating circumstance, contact your lab TAs and Instructor.

### (2 pts) Getting Set Up

These **exercises need to be completed individually by each student**. This is to ensure that everyone gets setup properly! You can certainly ask your friend(s) questions! virtually ☺

1. Log into TEACH to see if you have an ENGR account. **If you do not have an ENGR account**, then you will need to create one by clicking on the link at the bottom of the login page that says: [Create a new account \(Enable your Engineering resources\)](#).



2. **Log on to the ENGR server through a secure shell.** Depending on your operating system, you need to go to the instructions for [Windows](#) or [MacOS/Linux](#).

#### Windows Users:

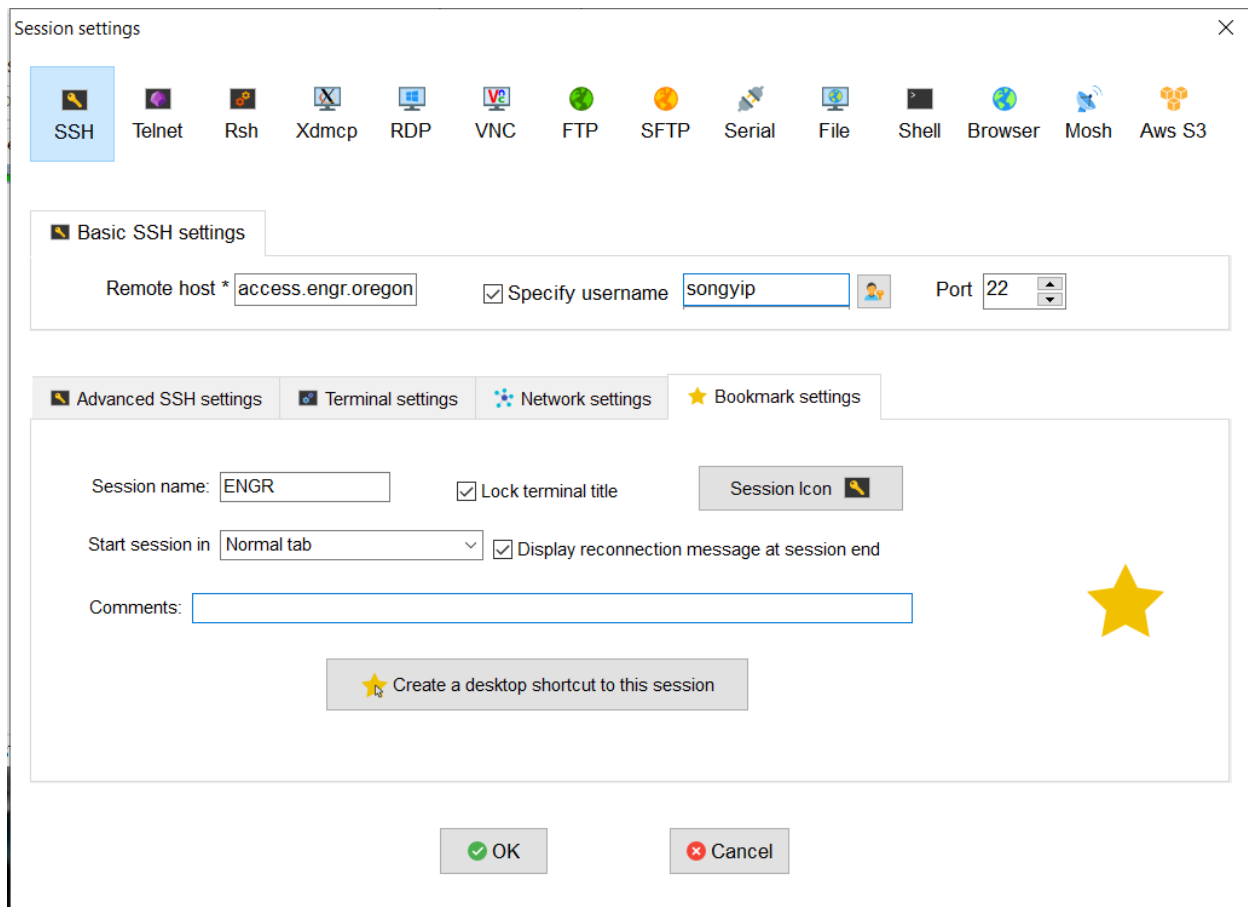
Download a free Secure Shell (ssh). We will use the MobaXterm this quarter.

MobaXterm: <http://mobaxterm.mobatek.net/download-home-edition.html> (Use Installer Edition, unless you are not using your own computer, and extract files in zip to not get a bunch of messages about accessing files.)

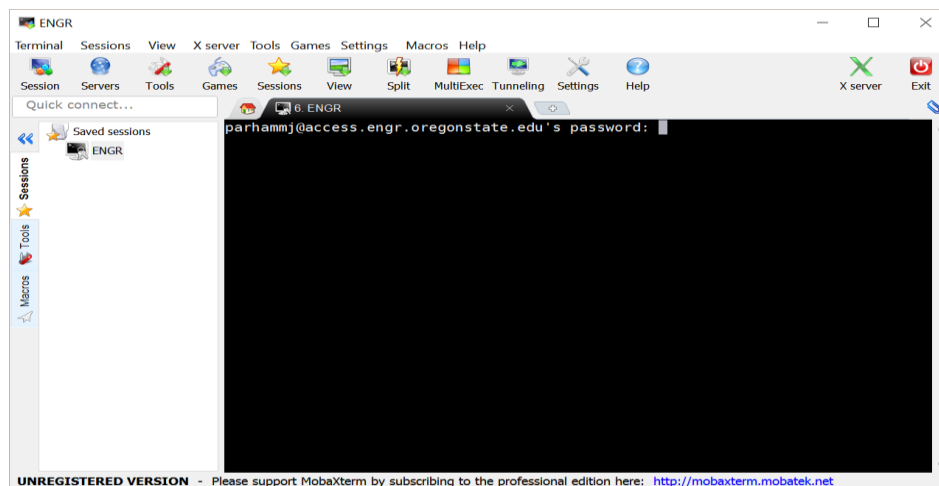
Once you have installed MobaXterm, open MobaXterm. Go to **Sessions -> New Session**, and click on the **SSH icon**.

Enter **access.engr.oregonstate.edu** as the remote host, and click on the **Specify Username** checkbox to enter your username in the appropriate field.

Click on the **Bookmark settings** tab to save your session with a specific name, e.g. ENGR. Lastly, click OK!!! ☺

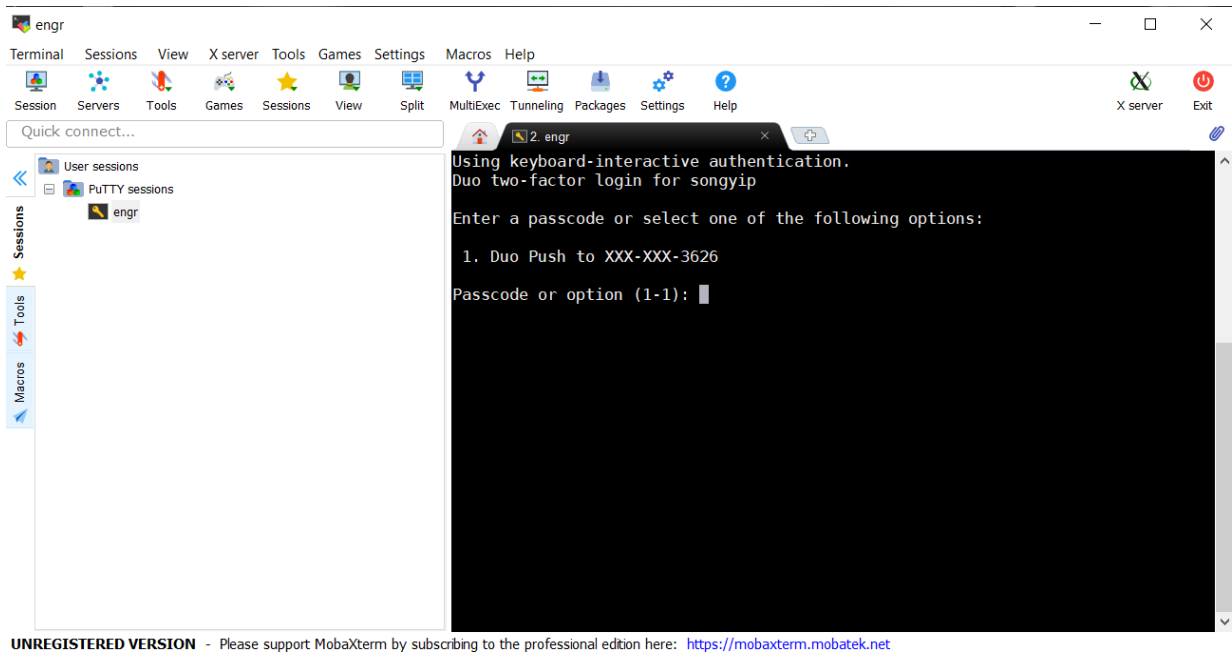


The ENGR session you just created will connect, and you will be asked to enter your password at the prompt. **\*\*Note:** You will not see anything as you type your password. This is a security feature in Linux!!!

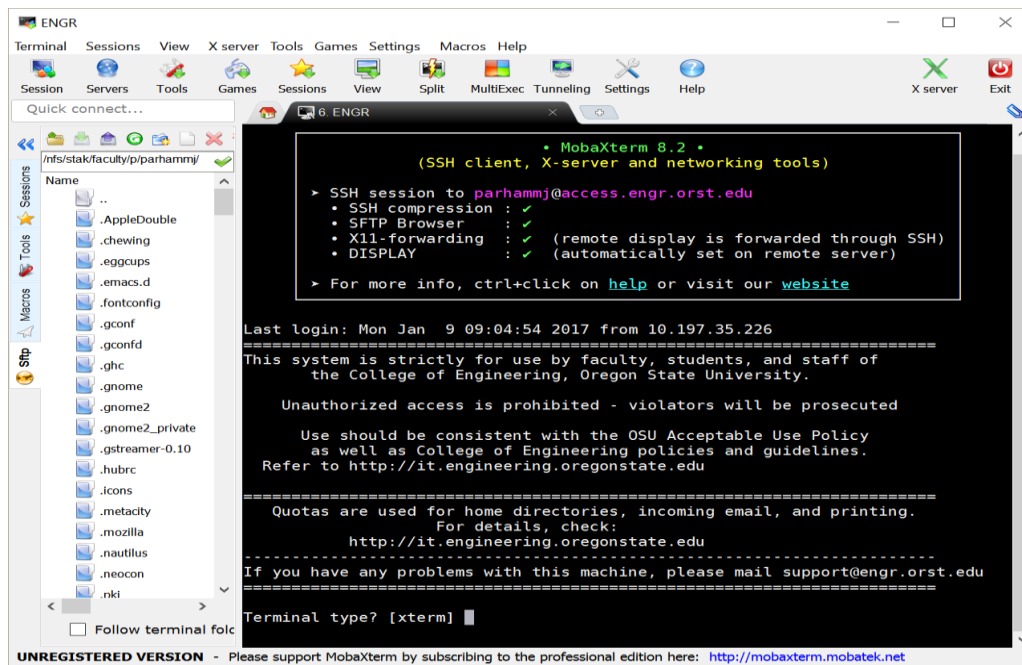


You will need an authentication for DUO Two-factor login, please read this [information](#) and sign up [here](#).

Or you can bypass the need for DUO by setting up SSH Keys, see instructions [here](#).



After a successful login, press **ENTER** at the prompt below:  
**Terminal type? [xterm]**



Then you get a prompt that looks something like this:  
**flip1 ~ 1%**

### MacOS or Linux Users:

You have a terminal w/ ssh built into the OS. Open the terminal by going to **Go -> Utilities -> Terminal** (on a Linux system, you might be able to click on the desktop and select Terminal). At the prompt, type **ssh username@access.engr.oregonstate.edu**. (replace **username** by your onid username)

The ENGR session you just created will connect, and you will be asked to enter your password at the prompt. **\*\*Note:** You will not see anything as you type your password. This is a security feature in Linux!!!

A terminal window titled "parhammj — ssh parhammj@access.engr.oregonstate.edu — 80x24". The output shows the last login time as "Thu Jan 3 08:11:30 on ttys000". The user "parhammj" is prompted for their password, indicated by a question mark icon. The prompt is "[Jennifers-MacBook-Air:~ parhammj\$ ssh parhammj@access.engr.oregonstate.edu parhammj@access.engr.oregonstate.edu's password: ?]".

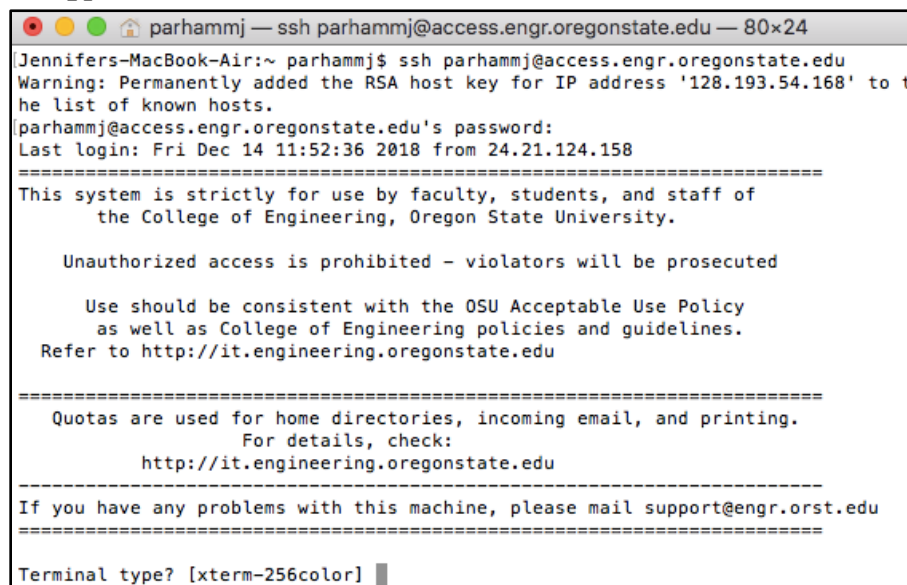
```
parhammj — ssh parhammj@access.engr.oregonstate.edu — 80x24
Last login: Thu Jan  3 08:11:30 on ttys000
[Jennifers-MacBook-Air:~ parhammj$ ssh parhammj@access.engr.oregonstate.edu
parhammj@access.engr.oregonstate.edu's password: ?]
```

You will need an authentication for DUO Two-factor login, please read this [information](#) and sign up [here](#).

Or you can bypass the need for DUO by setting up SSH Keys, see instructions [here](#).

After a successful logon, press enter at the prompt below:

**Terminal type? [xterm-256color]**

A terminal window titled "parhammj — ssh parhammj@access.engr.oregonstate.edu — 80x24". The output shows a warning about the RSA host key for IP address '128.193.54.168'. The user "parhammj" is prompted for their password. The last login time is "Fri Dec 14 11:52:36 2018 from 24.21.124.158". The terminal displays a series of system messages and prompts, including a warning about unauthorized access and a prompt to type the terminal type. The prompt is "Terminal type? [xterm-256color]".

```
parhammj — ssh parhammj@access.engr.oregonstate.edu — 80x24
[Jennifers-MacBook-Air:~ parhammj$ ssh parhammj@access.engr.oregonstate.edu
Warning: Permanently added the RSA host key for IP address '128.193.54.168' to t
he list of known hosts.
parhammj@access.engr.oregonstate.edu's password:
Last login: Fri Dec 14 11:52:36 2018 from 24.21.124.158
=====
This system is strictly for use by faculty, students, and staff of
the College of Engineering, Oregon State University.

Unauthorized access is prohibited - violators will be prosecuted

Use should be consistent with the OSU Acceptable Use Policy
as well as College of Engineering policies and guidelines.
Refer to http://it.engineering.oregonstate.edu

=====
Quotas are used for home directories, incoming email, and printing.
For details, check:
http://it.engineering.oregonstate.edu
=====
If you have any problems with this machine, please mail support@engr.orst.edu
=====
Terminal type? [xterm-256color] █
```

Then you get a prompt that looks something like this:

```
flip1 ~ 1%
```

Now, let's enable the use of your Backspace key, so that you don't get the pesky "^?" in your text editor.

- Go to your **Terminal Preferences** under the **Advanced** tab and enable "Delete Sends Ctrl-H".

### 3. Practice Linux Commands

- **Listing the contents of a directory/folder.**

At the prompt, type the following commands one by one to look at your files and directories in your home directory. Note the differences on a piece of paper.

```
ls
```

```
ls -a
```

```
ls -l
```

```
ls -al
```

**\*\*Note:** You should notice a `.` and `..` directory listed. The `.` directory refers to your current directory, and the `..` directory refers to one directory above your current directory. The `~` refers to your home directory.

- **Instructions for Unix/Linux commands.**

UNIX/Linux provides you manual pages for the commands. You are encouraged to read these manual pages when you have questions about a specific command or want more details about the options to use with a command. Use the **space bar** to scroll forward through the manual pages (one page at a time), **press b** to scroll backwards (one page at a time), and **press q** to quit the manual page. You can also use the up and down arrow keys to scroll forward and backward one line at a time, but who wants to do this 😊

```
man ls
```

- **Find Unix/Linux commands.**

In addition, if you are not sure what a command is in UNIX/Linux, then you can find the appropriate command using `apropos` and a keyword. For example, what are the UNIX/Linux commands for editing a file, working with a directory, etc.

```
apropos editor
```

```
apropos directory
```

You may have noticed that you get more text than what can fit in your terminal window. To view data a page at a time in your terminal, you can pipe the command contents through another command called `less`. This will allow you to scroll through the pages using **space bar, b, and q** just as you did with the manual pages.

```
apropos directory | less
```

- **Directories/Folders.**

Make a directory in your home directory named `labs`, and change into the `labs` directory.

```
mkdir labs
```

```
cd labs
```

Create a directory in your `labs` directory named `lab1`, and change into the `lab1` directory.

```
mkdir lab1
cd lab1
```

Make a note of your present working directory.

```
pwd
```

You can go back/up a directory by using two periods/dots together, and you can go back to your home directory by using the tilde, ~. Use the pwd to confirm you are back in your home directory.

```
cd ..
cd ~
pwd
```

Now, change into the labs/lab1 directory by using your **up arrow key** to take you through the history of commands you've used in the past. You should see the `cd labs` and `cd lab1` command you typed earlier. You can also change directly into the lab1 directory by using `cd labs/lab1`.

A good rule of thumb is to use `pwd` at any time to determine where you are, in case you forget. ☺ Also, don't be scared to use `ls` as often as you need to see a listing of your current directory!

- **Creating Files.**

Use the vim editor to create a C++ file containing your first C++ program.

```
vim hello.cpp
```

Below is an overview of vi/vim and a set of useful commands. **\*\*Remember you can `man vim` to see the manual pages for vi/vim.** In addition, I have provided a useful vim tutorial under **Useful Links on our Canvas website**. One of the default editors that come with the UNIX operating system is called vi (**v**isual editor) or vim (**v**isual editor **i**mproved). [Alternate editors for UNIX include emacs and pico.]

The UNIX vi editor is a full screen editor and has two modes of operation:

- **Command mode** commands which cause action to be taken on the file, and
- **Insert mode** in which entered text is inserted into the file.

In the insert mode, every character typed is added to the text in the file; pressing the <Esc> (*Escape*) key turns off the Insert mode.

**Basic Commands (in command mode):**

```
:w<Return> - write out modified file to file named in original invocation
:wq<Return> - quit vi, writing out modified file to file named in original invocation
:q<Return> - quit (or exit) vi
:q!<Return> - quit vi without saving the latest changes
:0<Return> - move cursor to first line in file
:n<Return> - move cursor to line n
:$<Return> - move cursor to last line in file
:set number<Return> - insert line numbers into vi file
```

**/search**<Return> - find first occurrence of search from the location of cursor in the downward direction

**?search**<Return> – find first occurrence of search from the location of cursor in the upward direction

**n** – move cursor to next occurrence of last search (**in direction of search**)

**j** [or down-arrow] - move cursor down one line

**k** [or up-arrow] move cursor up one line

**h** [or left-arrow] move cursor left one character

**l** [or right-arrow] move cursor right one character

**0** (zero) - move cursor to start of current line (the one with the cursor)

**\$** - move cursor to end of current line

**^** - move cursor to the first non-whitespace character in a line

**w** - move cursor to beginning of next word

**b** - move cursor back to beginning of preceding word

**u** – undo whatever you last did

**x** – delete current character

**dd** – delete current line

**yy** – yank line and put into buffer for pasting

**p** – paste text in buffer to line below cursor

**i** – enter insert mode and enter text before the cursor

**a** - enter insert mode and append text after cursor

**o** - enter insert mode and enter text on line below cursor

**<Esc>** - get out of insert mode and enter command mode

Write the infamous “hello world” program as your first piece of C++ code. Use the style guideline on the class website for suggestions on how to format your code:

[http://classes.engr.oregonstate.edu/eecs/fall2019/cs161-001//161\\_style\\_guideline.pdf](http://classes.engr.oregonstate.edu/eecs/fall2019/cs161-001//161_style_guideline.pdf)

```
#include <iostream>

int main() {

    std::cout << "Hello World!" << std::endl;

    return 0;
}
```

- **Compiling/Executing C++ Code.**

Now, we will use the GNU C++ compilers on the server to compile your C++ “hello world” program. Then, you will execute the compiled program.

```
g++ hello.cpp -o hello
```

```
./hello
```

You can also compile/execute C++ code using Clang compiler:

```
clang++ hello.cpp -o hello
./hello
```

#### (4 pts) Answer Reflective Metacognitive Questions

Record your answers to the following questions in a word or text document on your computer to submit at the end of lab. For each question, make a change to the hello.cpp file, recompile (**g++** or **clang++**), and possibly execute the program (**./hello**) if there are no errors from the compilation.

**Question 1:** What happens when you forget the semicolon at the end of the cout statement? Be specific.

```
std::cout << "Hello World!" << std::endl
```

**Question 2:** What happens when you forget to include iostream, i.e. remove **#include <iostream>** from the program? Be specific.

**Question 3:** What happens when you put **"Hello World"** on a new line? Be specific.

```
std::cout <<
    "Hello World!" << std::endl;
```

**Question 4:** What happens when you put **World!"** on a new line? Be specific.

```
std::cout << "Hello
    World!" << std::endl;
```

**Question 5:** What happens when you remove the **<< std::endl**? What is the difference in having it and not having it?

```
std::cout << "Hello World!";
```

**Question 6:** What happens if you forget to include the main function? Note: Instead of removing the lines from the file, comment them out using **//**. Put a **//** in front of the lines with **int main() {, return 0;, and }**. Be specific.

#### (2 pt) Add a variable and reading input to the program:

Now, let's add a variable (or a place to store information) in our program. We will create a place to store an integer (or whole number). Then we will ask/prompt the user to enter a number, read the input from the user, and display it back to the screen.

```
#include <iostream>
```

```
int main() {
    int number;
```



```

    std::cout << "Hello World!" << std::endl;

    std::cout << "Enter an integer: ";
    std::cin >> number;
    std::cout << number << std::endl;

    return 0;
}

```

Compile and execute your C++ “hello world” program again with the new changes.

```

g++ hello.cpp -o hello
./hello

```

For each question, make a change to the hello.cpp file, recompile (**g++** or **clang++**), and possibly execute the program (**./hello**) if there are no errors from the compilation.

**Question 7:** What happens if you mix the direction of >> with the **cin**? Be specific.

```
std::cin << number;
```

**Question 8:** Why do you need to create a variable to display a number entered by the user?

### **(2 pts) Upload Written Document (.doc or .txt) and Program (.cpp) to [TEACH](#)**

Since you have to upload your assignments from your local computer or the ENGR server to TEACH, then we should practice now☺ It is easy to upload files from your local computer to TEACH, as long as you remember where you save it!!! ☺ However, it is not as easy for you to upload files from the ENGR server to TEACH.

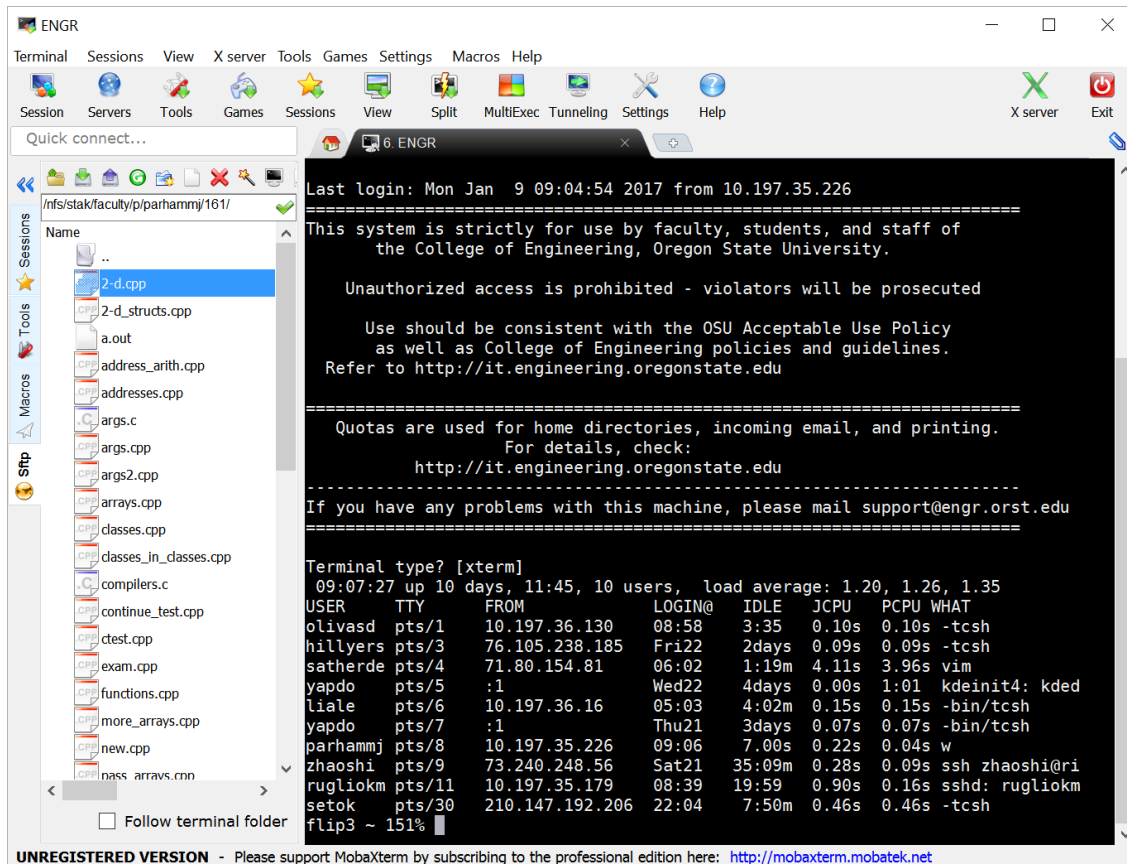
#### **Two ways to upload the file from ENGR to TEACH.**

- **Transfer Files from ENGR to local computer**

You can transfer the file to your own computer, then upload it to TEACH:

**Windows Users:** MobaXterm already has a sftp client installed. Notice the window pane on the left with folders and files. You can drag and drop a file to your local desktop to select on TEACH.

**MacOS Users:** (See “Map Network Drive” section below)



OR

- **Map Network Drive**

You can map a network drive and choose the file from the mapped network drive. This allows you to directly work off the server as if it were a disk drive on your computer. You can follow these instructions to map a network drive for Windows or MacOS.

**Windows:**

<http://it.engineering.oregonstate.edu/accessing-engineering-file-space-using-windows-file-sharing>

**MacOS:**

<http://it.engineering.oregonstate.edu/accessing-engineering-file-space-using-os-x-smb-command>

**If you want to use the drive off campus,** then you must download the **Cisco VPN Client** from OSU: <http://oregonstate.edu/helpdocs/osu-applications/offered-apps/virtual-private-network-vpn>

**\*\*NOTE:** You may want to put these programs on a flash drive to carry with you in your backpack. This will help you get around from any computer without needing your laptop all the time.