LAB #2 –Linux, vim, & C++

1. Open your secure shell (ssh) client and connect to: access.engr.oregonstate.edu. Refer back to Lab #1 if you don't quite remember how to do this.

2. In Linux, every command has a manual page, which provides you information on what the command means, how to use the command, and a description of the available options for a command. Linux commands do not have spaces in them and they are lower case. This is important because **Linux is case sensitive**!!! Following a Linux command is a space followed by arguments. Some arguments may be required and some are optional such as options, which are preceded by a dash.

   Linux_command –option required

   If an argument is optional in Linux, then it is enclosed in brackets, [], and required arguments do not have brackets. For example, **man ls**, and notice that everything supplied to ls is optional. You can also use the command and --help to get a brief manual page for the command, i.e. **ls --help**

3. In order to get more familiar with the Linux/UNIX environment, you will do a few Linux-type exercises at the beginning of each lab. Today, we will learn the copy, move, and remove commands, i.e. **cp**, **mv**, and **rm**. First, look at the manual page for each of these commands. Remember to use the **space bar**, **b**, and **q** for moving around in the manual pages.
   man cp
   man mv
   man rm

4. First, let's play with these new commands before moving forward. Copy your hello.cpp program from the labs/lab1 directory to your home directory, **cp labs/lab1/hello.cpp ~**. This says, copy the hello.cpp file located in the labs/lab1 directory into my home directory. Use **ls** to list the directory contents and make sure you have a hello.cpp file in your home directory.

5. Now, rename the file to hello2.cpp by using the move command, **mv hello.cpp hello2.cpp**. Use **ls** to list the directory contents and make sure you no longer have a hello.cpp file and you now have a hello2.cpp file.

6. Create a test directory, and then change into that directory.
   mkdir test
   cd test

7. Copy the hello2.cpp file from your home directory to the test directory you are currently in. Remember that **..** is up/back a directory. You could also say **mv ~/hello2.cpp** because you know that hello2.cpp is in your home directory.
   cp ../hello2.cpp .

8. Now, go back to your home directory or up/back a directory, and remove the file hello2.cpp file in your home directory and remove the test directory and its contents, which contains the file hello2.cpp. Use **ls** to make sure you see the hello2.cpp file and the test directory in your home directory. Type in these commands:
   cd ..
   ls
   rm hello2.cpp (when prompted press **n** so you don't remove it)
   rm –f hello2.cpp (notice no prompt, -f forcefully removes without asking)
   rm test (notice it won't remove a directory, even with -f)
   rm –r test (notice the prompt, -r recursively descends into a directory to remove it and its contents, note you can use –rf together to avoid all the prompts)
   ls (you shouldn't see hello2.cpp or test)

9. Change into your labs directory, create a lab2 directory, and then change into that directory. DO NOT use spaces in directory or file names in Linux.
   cd labs
   mkdir lab2
   cd lab2

10. There are a few shortcuts in Linux that you want to be familiar with using. One is the use of up/down arrows to move through your history of commands. At the shell prompt, press the **up arrow** and note what happens, and then press the **down arrow** and note what happens.

11. Another useful shortcut is **tab completion**. Go up two directories with **cd ../..**, and then let's change back into the labs directory. This time, after typing **cd** and **l**, then press the tab key. This will complete your labs word because it is the only option in your home directory that starts with an l. Now, try changing into the lab2 directory again using tab completion, but this time you'll be presented with two options that start with an l.

12. Now for some programming! We'll practice incremental programming so you can learn good habits. Increment programming simply is building your program in stages. This limits the amount of code you need to review if (when) you have bugs. You'll write a program that rolls a die. Note- many people aren't aware that dice is plural. ☺ We don't have the tools yet to roll more than one die. Maybe later.

    What will you need? Read the number of sides for the die. Use the random number generator function to simulate the roll. Then display the results. This is good design practice. Identify inputs and expected outputs. Then figure out how you turn the first into the last. You also need to figure out what type of data you'll be using so you

know what data type to assign to all variables (and functions when you start writing your own).

Start writing your program with the main() function. Put in a cin to read a value and a cout to print it to the screen. Did you remember to output a new line? Did you provide a user prompt to explain what was expected, such as the number and types of input? Once that is all working we move on to actually 'rolling the die'.

Figure out how you need to call the random function to simulate a roll of a die. You need to limit the result to integers that range from 1 to the number of sides. And yes, 1-sided is possible. (There's a die you can order online that's a Mobius strip with a number 1 printed on it.)