

LAB #2 – Linux Commands/Run Python

Many students get confused about the differences between MobaXterm, vim, and python3.

MobaXterm: just lets you access that machine in Kelley without being in front of it.

vim: editor on the ENGR server, just like notepad, word, etc...

python3: interpreter for python programs on the ENGR server

An advantage to our small class/lab is that we can all help one another easily. You are always more than welcome to **ask for help** from friends in this lab, and you are always more than welcome to **offer help** to friends!!! 😊

(2 pts) Linux File Manipulation:

For this lab, the **exercises on the computer need to be completed by each student** in the lab. This is to ensure that all students understand the fundamentals of Linux and vim used throughout the rest of the quarter!!!

1. Now, open your secure shell (ssh) client and connect to:
access.engr.oregonstate.edu
2. This opens a terminal window, which is similar to a Windows command window that presents you with a DOS prompt, but instead of being on your local Windows machine, you are on a remote Linux machine located somewhere else. This is why you are prompted for a username and password to get into the machine because we don't want just anyone on the ENGR machines!!! Once logged into the machine, you get a prompt similar to the DOS prompt, where you can type commands directly to the operating system. 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.py program from the labs directory to your home directory, **cp labs/hello.py ~**. This says, copy the hello.py file located in the labs directory into my home directory (denoted by ~). Use **ls** to list the directory contents and make sure you have a hello.py file in your home directory.
5. Now, rename the file to hello2.py by using the move command, **mv hello.py hello2.py**. Use **ls** to list the directory contents and make sure you no longer have a hello.py file and you now have a hello2.py file.
6. Create a test directory, and then change into that directory.

```
mkdir test
cd test
```
7. Copy the hello2.py 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.py . because you know that hello2.py is in your home directory.

```
cp ../hello2.py .
```
8. Now, go back to your home directory or up/back a directory, and remove the file hello2.py file in your home directory and remove the test directory and its contents, which contains the file hello2.py. Use ls to make sure you see the hello2.py file and the test directory in your home directory.

```
cd ..
ls
rm hello2.py (when prompted press n so you don't remove it)
rm -f hello2.py (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, also rmdir will remove a directory)
ls (you shouldn't see hello2.py 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.

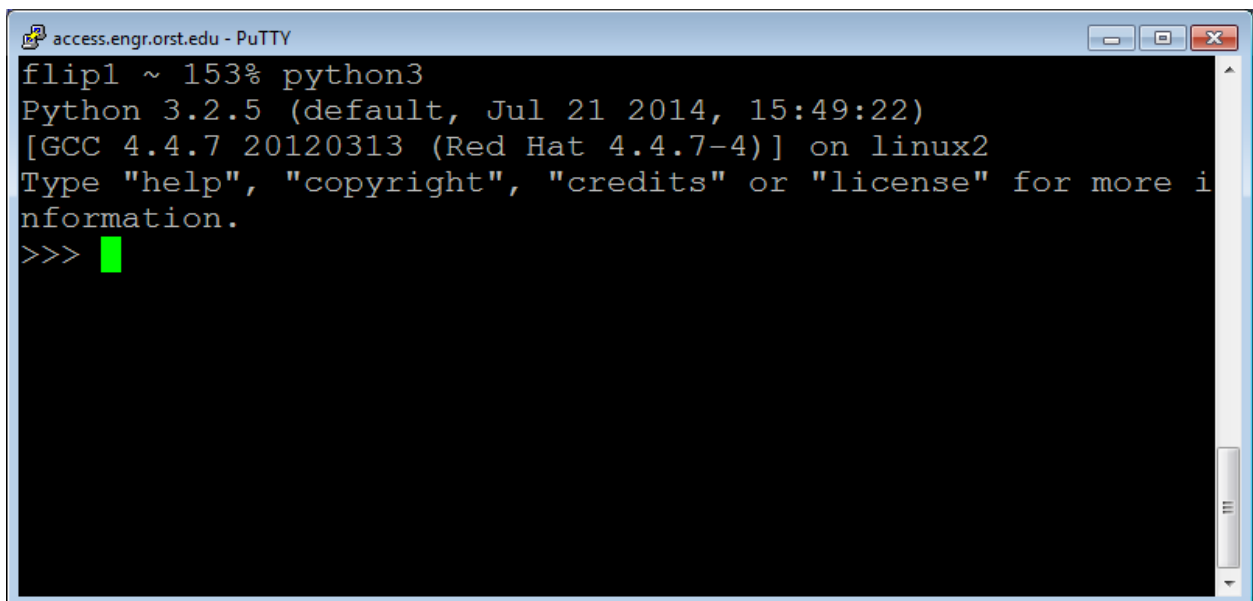
```
cd labs
mkdir lab2
cd lab2
```

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. You can view the contents of the file by using the command **more** or **less**.
more hello.py

(3 pts) More python and vim practice

You have already written a small "hello world" python program in Lab #1. In this lab, we will learn a little more about interpreters and writing python code.

1. To begin, we are going to execute a few python statements to understand the semantics and syntax before writing a full program. Let's enter the following examples for you to see what python syntax is. First, Python statements do not need to be ended by a semicolon, which is ignored if supplied. However, it might be a good habit to get into because other languages require it, and it will be required in CS 161!☺ Let's start the Python interpreter by just typing **python3** at the command prompt to start the python interpreter.



```
access.engr.orst.edu - PuTTY
flip1 ~ 153% python3
Python 3.2.5 (default, Jul 21 2014, 15:49:22)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-4)] on linux2
Type "help", "copyright", "credits" or "license" for more i
nformation.
>>> █
```

2. Instead of writing a program and running the program through the interpreter, let's just run single commands directly through the interpreter. For example, print a message for the user to read. Strings of text have to be surrounded by quotes.

```
print("Practicing math ops and concatenation");
```

3. Now, let's try out some **math operators** such as addition, subtraction, division, multiplication, exponent, and mod/remainder:

```
print(10 + 5);  
print(10 - 5);  
print(10 / 5);  
print(10 * 5);  
print(10 ** 5);  
print(10 % 5);
```

4. In python3, they give you a choice between a divide that produces a real/floating point number and one that produces an integer number. Try entering in the following statement, and note the differences you see between the previous division:

```
print(10 // 5);
```

Why do you think they provide two divisions? How could be useful?

5. What if we want to **concatenate information together**? We use the + symbol to concatenate (or put together) strings of information. For example, you want the message "We have almost 300 new students interested in CS this year! Woohoo!!!" to appear on the screen, but you want to print the number separate from the text. See what happens when you enter the statement below:

```
print("We have " + 20 + " honors students interested in CS this year!");
```

Notice you get an error! We have to convert the numbers to strings to put them together with strings in a print statement. This is because the + symbol can also be used between numbers to mean addition. It depends on the context you are using the + symbol. We can temporarily change the type of data using typecasting. In this case we need the numbers to act as strings of text. We use a function called str() to do this.

```
print("We have " + str(20) + " honors students interested in CS this year!");
```

6. To exit the interpreter, just type **exit()**;
7. Let's put it all together in an example program. Open a new file by typing:

```
vim math_ex.py
```

```
print("Practicing math ops and concatenation");  
print(10 + 5);  
print(10 - 5);  
print(10 / 5);  
print(10 * 5);
```

```
print(10 ** 5);
print(10 % 5);
print(10 // 5);
```

```
print("We have " + str(20) + " honors students interested in CS this year!");
```

Now run your program by typing `python3 math_ex.py`.

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 class 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

Here is a Linux and vim cheat sheet to help you reference some of these commands quickly. <http://classes.engr.oregonstate.edu/eecs/fall2016/cs160h-001/labs/CheatSheet.pdf>

You can find more Linux and vim cheat sheets and tutorials on the links page of our class website: <http://classes.engr.oregonstate.edu/eecs/fall2016/cs160h-001/links.html>

(5 pts) Program a Larger Equation

In class, we wrote an equation for determining the largest and smallest signed number and the largest unsigned number given one byte. Let's write this same equation in python to test it out!

1. First, let's just make sure you can program the equations for one byte.
2. What happens when we want to calculate this for x bytes?
 - **As a class, let's write the algorithm/step-by-step instructions for this activity.**
3. Now, let's program this in python3!!!
 - How do we get the value for x?
 - What happens when we try to use x in our equations?
 - How is this similar to the error we received above?
 - What do you think we can do to eliminate this error?

Make sure you sign-up with a TA for demoing/explaining your Assignment #2 in week 2. The doodle polls are listed on the course home page beside the TA office hours: <http://classes.engr.oregonstate.edu/eecs/fall2017/cs160h-001/> **You are penalized for failure to schedule an appointment within the week or missing a scheduled appointment.**