CSCI 141 Computational Problem Solving
Lab 2: Command Line Crash Course

The goal of this lab is to become comfortable using a subset of common Linux commands, writing Python programs in a text editor, and executing Python files from the command line.

Linux commands are shown in **`bold Courier font.`** Text you see in commands that is in brackets and italicized like ***`<filename>`*** is meant to be replaced with text specific to the situation, i.e. you'd replace filename with the actual name of a file. Python code is indicated by `non-bolded Courier font`. Important terms are indicated by **blue highlighting**.

Part One: Using the command line

If you are reading this, then you successfully downloaded the .zip file containing materials for this lab. Make sure to download and extract the .zip file in a location you can easily find.

This section of the lab will introduce (or review) some useful Linux commands as well as the directory structure and how to navigate it.

Open a terminal and type:

```
pwd
```

This will show you the path of the **current working directory**, that is, where you are located in the file system of the computer. Type the command:

```
cd
```

This will return you to your home directory. You could also accomplish this by typing **`cd ~`** or **`cd ~/`**

The ~ is shorthand for your home directory.

You can see the contents of your home directory by typing:

```
ls
```

If you downloaded and extracted the Lab2 files properly, you should see a folder called Lab2 as well as the Lab2.zip file you downloaded. Let's create a new directory for lab materials:

```
mkdir CSCI141Labs
```

The command mkdir tells the system to make a directory with the name that follows:

```
mkdir <name_of_directory>
```

If you enter:

```
ls
```

you will see CSCI141Labs listed now as one of the items in your home directory. Enter the CSCI141Labs directory:

`cd CSCI141Labs`

The command cd tells the system to change directories. Using cd with no arguments returns you to your home directory, otherwise, cd will take you to the directory you specify:

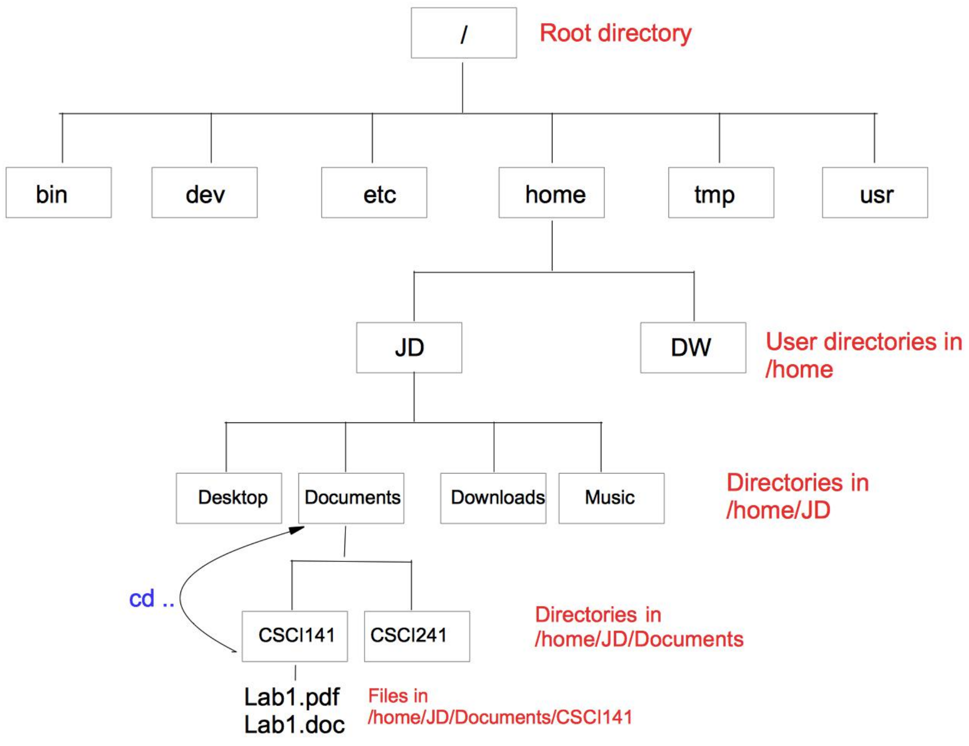
`cd` #takes you home

`cd` ***`<name_of_directory>`***

If you use the command **`pwd`**, you will see that now you are in the CSCI141Labs directory. To get back to your home directory, you could enter **`cd`**, **`cd ~`**, or:

`cd ..`

The **`..`** means to go up one level in the **directory tree** - but what is the directory tree?



The figure above shows an example of the directory tree structure for a Linux file system. This is analogous to what you see if you look in “My Computer” on a Windows system or “Finder” on a Mac. Each box is a directory (you can think of this as a folder), and may or may not contain subdirectories.

Here, we see that within the home directory, there are two user directories, JD and DW. User JD has four directories in their home folder. The Documents sub-directory has two folders. For one of these, two files are listed.

The command **`cd ..`** takes you up one level in the tree. So, for example, if I am currently in:

/home/JD/Documents/CSCI141

and then we type:

```
cd ..
pwd
```

We will see that the current working directory is:

/home/JD/Documents

If we do this again:

```
cd ..
pwd
```

The current working directory is:

/home/JD

You can access a directory by providing its **absolute path**, that is, the list of all of the directories that contain it. For example, if we wanted to navigate to the CSCI241 directory, we could type:

```
cd /home/JD/Documents/CSCI241
```

OR

```
cd ~/Documents/CSCI241
```

Remember that ~ is equivalent to /home/JD in this case. This will work from any location (provided we have permission to access the directory). You can also access a directory using its **relative path**, that is, its location relative to your current location. Suppose we are in the directory Documents. To get to the CSCI241 folder, we could type:

```
cd CSCI241
```

Since we are already in /home/JD/Documents, we do not need to include those parts of the absolute path to navigate to CSCI241. Similarly, if we were in /home/JD, we could get there with:

```
cd Documents/CSCI241
```

Be careful! If we try to navigate to CSCI241 from /home/JD/Downloads, we would need to use the absolute path, or something like:

```
cd ../Documents/CSCI241
```

The .. says to go up one level to /home/JD, after that, we can enter /Documents/CSCI241.

Suppose you are the user JD and are currently working in the directory /home/JD/Desktop and you want to access the file program.py. The file extension .py indicates that this is a Python file. You can navigate to the directory that contains it by typing:

```
cd /home/JD/Documents/CSCI141
```

OR

```
cd ~/Documents/CSCI141
```

OR

```
cd ../Documents/CSCI141
```

Then, you could execute the python file by typing:

```
python program.py
```

When you run python programs from the command line you will need to be aware of their location in the file system.

Let’s finish with the example directory tree, and get back to your home directory.

To move files and/or directories from one directory to another, use the command **mv**. For example, let’s move the Lab2.zip file from your home directory to the CSCI141Labs directory:

```
cd #return to home directory
```

```
mv Lab2.zip CSCI141Labs
```

Now, you can **cd** into the CSCI141Labs directory and verify with ls that the file is present. In general, the syntax for mv is:

mv ***<source> <destination>***

This moves a file from the source directory to the destination directory. You can change the name of a file by moving it to a different file in the same directory. Give it a try:

```
mv Lab2.zip Lab2Files.zip
```

Remember – you must be in the directory containing the file to do this using the command above.

You can copy a file from one location to another using the command **cp**:

cp ***<source> <destination>***

To copy a directory, you need to add the option –r which indicates recursive copy, that is, to copy the directory and everything inside of it, including subdirectories and their contents. Copy the Lab2 directory from your home directory into the CSCI141Labs directory:

```
cd
```

```
cp –r Lab2 CSCI141Labs
```

When we copy using the syntax above, it creates a new directory in CSCI141Labs called Lab2. If we were to copy using:

```
cp –r Lab2/ CSCI141Labs
```

it will copy the files in the Lab2 directory into CSCI141Labs without recreating the Lab2 directory.

Make sure that you have moved the Lab2 files into your CSCI141Labs directory, continue with Part two.

Part Two: Debugging and Executing .py files

In this section of the lab, you will debug a python program, then execute it from the command line with a set of given inputs to demonstrate correctness.

Marathon times

Open the file marathon.py in a text editor (gedit on the lab computers). You can do this from the command line by navigating to the Lab2 directory and typing:

```
gedit marathon.py
```

Be sure that syntax highlighting for Python is selected.

Results of shorter distance races are often used to predict finishing times for marathons. One such formula computes the pace per mile for a marathon as 1.3 times the pace per mile. The program in marathon.py uses this formula to calculate the predicted marathon finish time based on an input 5K (3.1 mile) race time in minutes. The input to the program should be a 5k finishing time in minutes (assume it is non-negative), and the output is the predicted marathon finishing time in the following format:

```
Predicted marathon finish time is 5 hours 30 minutes.
```

The number of hours is an integer, and the number of minutes is rounded to a whole number. For example, if the program calculates a marathon finishing time of 255.75 minutes, the output should read:

```
Predicted marathon finish time is 4 hours 16 minutes.
```

You can round a floating point value in Python using the function round:

```
round(15.75) = 16
```

The code presented in marathon.py has several errors, syntactical and otherwise. If you try to run the program as is, it will exit with an error. Open a new terminal, navigate to the Lab2 directory using cd, and type:

```
python marathon.py
```

The output should look like this:

```
  File "marathon.py", line 8
    5ktime = input("Enter 5k finishing time (in minutes):")
         ^
SyntaxError: invalid syntax
```

Correct the code in marathon.py to produce a working program according to the specifications. You should make use of the python interpreter to run the program and help you locate errors. To verify that your program is working correctly, you will run the small set of **test cases** given below. For each test case, we know what the output should be for a given input, and test that the program produces the correct output.

Input:	50	Expected output: Predicted marathon finish time is 9 hours 9 minutes.
Input:	5	Expected output: Predicted marathon finish time is 0 hours 55 minutes.
Input:	27.31	Expected output: Predicted marathon finish time is 5 hours 0 minutes.
Input:	19.9	Expected output: Predicted marathon finish time is 3 hours 39 minutes.
Input:	0	Expected output: Predicted marathon finish time is 0 hours 0 minutes.