CSCI 204 – Introduction to Computer Science II

Lab 1 – Linux and Python

1. Lab Objectives

- Learn to use the Linux command line
- Become familiar with commands: `date, ls, mkdir, cd, pwd, &, cp, rm,` and `mv`
- Review Python you learned from CSCI 203

2. Introduction

We will exercise two subjects in this lab. One is to learn a few more Linux commands, the other is to learn a few new Python commands and refresh what you already know. First we learn a few more Linux commands.

The lab work in CSCI 204 will all be individual work. While discussion is encouraged, you must code yourself.

3. More Linux commands

Although Linux uses a window manager to give it a GUI interface similar to Windows, a lot of functionality is best available via a **shell window** with a **command line**, where one can direct actions using a text command.

- A **shell** is a language which communicates with the operating system. It is an interface between the users and the operating system and the computer resources the operating systems manages.
- A **shell window** is a window with a prompt at which a user can issue commands to the operating system in text. Commands typed into this window are carried out by the operating system. The shell language includes a list of the commands the operating system will take.
- A **command line** is a text prompt where you can type shell commands.

You will need to use shell commands for this course and for many future computer science courses you will take.

If you have learned it elsewhere then you may know this material already. You still need to do the lab and turn in the answers. You may also want to help others who are new to the command line environment.

1. Opening a shell window

To open a shell window, right click on an open area on your desktop and choose "Open Terminal" from the pull-down menu. A window will open with a white screen and the word "Terminal" on the top bar. At the top of the white screen, you should see a prompt, and it should look like:

```
[yourUserName@computerName ~]$
```

Click the mouse point on the shell window to make it *active*. Note that if the initial starting directory is your *Desktop*, in which case, you'd see your prompt similar to the following,

```
[yourUserName@computerName Desktop]$
```

you can simply issue the command `cd` (that is, changing directory without a parameter) to bring you back to your home directory. You are able to issue command to the computer using text. For example, type the following command followed by the `Enter` key on your terminal.

```
date
```

You should see what you typed after the prompt and you should see the result of the command after you hit the `Enter` key. In this example, you should see a line similar to the following in your terminal.

```
Mon Jul 31 10:05:32 EDT 2017
```

2. Shell commands

Here is a list of the basic terms and commands you will need. We'll cover how to use them in the rest of this lab. Note that Linux uses the term *dictionary* to mean *folder*.

- A **path** is a list of folders and subfolders where something resides. For example, your Linux account is located at the path `/home/accounts/student/`. The Windows operating system also uses paths, e.g., `C:\Windows\ApplicationData\Mozilla\Firefox`.
- A **prompt** is a symbol or set of characters where you type commands. By default, your prompt displays your username, the computer you are on, and your current folder so it may look similar to the following
  `[abc123@computer ~/csci204]`
- The **gedit** command starts a simple text editor in a new window.
- The **&** command runs another command in a background mode where you can continue using the current shell window.
- The **pwd** command (*print working directory*) displays the name and path of the current folder.
- The **ls** command (*listing*) displays the contents of the current folder.
- The **mkdir** command creates new folders.
- The **cd** command (*changing directory*) changes from one folder to another.
- The **cp** command copies files and folders.
- The **mv** command moves and/or renames files and folders.
- The **rm** command deletes (removes) files and folders.

You might be wondering that why we use text commands anyway now that we have a nice windows system and a point-and-click mechanism. Think of this issue in two perspectives. One is that consider how many different icons you may have on a window system for you to click and how many different commands you can have with text. You can issue almost unlimited different commands in one text window! The second is that even you may have many, many icons or pull-down menus to work with in a windows system, you may have to hold and click many levels before your final command. For example, to go to a particular folder (or directory) under a Linux system with a windows system, e.g., your lab 9 work from CSCI 203, you may have to point and click a number of times before reaching there from your current CSCI 204 lab 01 directory, but with command line, you can issue one single command similar to the following to achieve the goal.

```
cd ~/csci203/labs/lab09
```

3. Creating a text file for submission

Begin editing a *lab01.txt* file for today's lab, which will be handed in, using **gedit** by typing

```
gedit &
```

at your prompt and hitting the `Enter` key. Note that the `&` key means running `gedit` in background so you still have the command window. Put the following information at the top of the file. Replace the student and instructor name with proper ones.

```
CSCI 204 Lab 01 Linux and Python
Lab section: CSCI 204.L61, Tuesday 10-11:50
Student name: Sam Snoopy
Instructor name: Professor Garfield
```

Number the problems you answer in this file using the following format.

```
Problem 1 answers: here goes my answer to the problem …
```

You will be asked to answer a number of questions and place the answers in the file `lab01.txt` you just created. Throughout the exercises, feel free to use the *copy and paste* feature that is available at the command prompt.

Problem 1: What is the content of your **prompt**?

Save your new file as *lab01.txt*.

If you know another text editor (`emacs`, `vi`, etc.) feel free to use it as long as you can launch it from the command line!

If you remembered to type the **&**, you will still be able to type in the shell window. If not, type **Ctrl-z bg** by holding down the **Ctrl** (Control) key and hitting **z**. Then type **bg** (for *background*) and hit `Enter`. The command sequence `Ctrl-z bg` sends the last command issued to the background so you can get the prompt back to type other commands.

4. Displaying your folder path

When you opened the shell window, you started out in your **home** folder. It's the base folder for your account. Windows uses the same concept. If somehow you have moved away from your home folder, you can use the Linux command `cd` without any argument to get back to the home folder.

You can tell you are in your home folder because your prompt shows a **tilde** (~) for your current folder name. The tilde is just a shortcut for "one's home folder." The path to your **home** folder is actually much longer. On Windows, you can get to your *MyDocuments* folder using a shortcut or navigate to `C:\Documents and Settings\user name\MyDocuments` using the full path. To see the full path to your current folder on Linux, type `pwd` at your prompt and hit the `Enter` key.

Problem 2: What is the full path to your home folder? (Use `pwd` to find out).

5. Displaying the contents of a folder

To see the contents of the current folder, type `ls` at your prompt and hit the Enter key. (That's a lowercase letter *l* in the command `ls`). You should see your *lab01.txt* file and some files that you created in CSCI 203. If coloring is turned on, files and folders will be shown in different colors. Folder names are usually followed by a / (forward slash) symbol.

Problem 3: What files and folders are inside your home folder? (Use `ls` to find out).

6. Creating and entering folders

Let's create a folder for this course. We want the new folder *csci204* under your home folder. To create the folder under the home directory, type the following command, assuming you are at your home directory.

```
mkdir csci204
```

follow the command by the `Enter` key. To get into this folder, type `cd csci204` at your prompt and hit the `Enter` key. Your prompt should now show that you are in the *csci204* folder.

Problem 4: What is the full path to your *csci204* folder? (Use `pwd` to find out.).

To go back up a folder, type `cd ..` (that is the command **cd** followed by two dots) at your prompt and hit the `Enter` key. You can also go into a folder from any other folder by using `cd` and its path (full or shortcut). The shortcut path to your *csci204* folder is *~/csci204*. You can get back to your home folder in one command by typing `cd ~` at your prompt and hitting the `Enter` key (typing plain **cd** will work too).

7. Copying, moving, and renaming files

Files and folders can be copied, moved, and renamed as needed. We'll make a copy of your *lab01 .txt* file. Go into your *home* folder. Then type `cp lab01.txt copy.txt` at your prompt and hit the `Enter` key. Use `ls` to see that the *copy.txt* file now exists. You can open it using the command `gedit copy.txt` to make sure it is really copied.

You can move a file from one folder to another using the **mv** command. To move *copy.txt* into your *csci204* folder, type `mv copy.txt csci204/` at your prompt and hit the `Enter` key. This moved the *copy.txt* file from the current folder (your home folder) into the *csci204 folder*. (Use **ls** and **cd** to see that the file has moved).

You can also rename files using the same **mv** command. To rename *copy.txt* to *extra.txt*, type `mv copy.txt extra.txt` at your prompt and hit the `Enter` key. (Use **ls** to see that the file has been renamed).

These commands can be combined and can be used locally as seen here or with paths when you want to move a file in some other folder. Go back to your home directory. Type

```
mv ~/csci204/extra.txt copy.txt
```

at your prompt and hit the `Enter` key. Notice that the file has moved back into your home folder and been renamed back to *copy.txt*. (Use **ls** to verify this).

Problem 5: What is the command to move your *lab01.txt* into your *csci204* folder? Quit editing your *lab01.txt* file first. Then move the file, edit *lab01.txt* again, and enter the command you used.

The **cp** and **mv** commands will also copy, move, and rename folders.

8. Deleting files

Your *copy.txt* file should be currently found in your home folder. Go into your home folder. To delete *copy.txt*, type `rm copy.txt` at your prompt and hit the `Enter` key. You should be asked if you really want to delete the file. Type `y` or `yes` and hit `Enter`. You can also use shortcut paths with the delete (**rm**) command.

Problem 6: Using a single command, how would you delete *copy.txt* if you aren't currently in your home directory (for example, what if you were in your *csci203* directory)? Hint: use the home shortcut ~ (the tilde key)

To delete a folder, you have to recursively delete all files and subfolders inside the folder. To test this out, first use **mkdir** to create a folder named *extra*. To delete this folder, type `rm -r extra` at your prompt and hit the Enter key. Again, you should be asked if you really want to delete it. (The *-r* option after the command **rm** stands for recursion.)

9. **Undeleting files**

If you accidentally delete a file or folder, it is gone. However, if the file existed several hours ago, there is a good chance you can get it back. Bucknell runs a program called *snapshot* on the Linux system. *Snapshot* makes a copy of all of your files every four hours. These copies are saved for a few days.

If you delete a file or folder and want it back (or mess it up and want to turn back the clock), go to where the problem is. Type `cd .snapshot`. If you do an `ls`, you will see folders with dates and times that look similar to the following.

```
daily.2017-07-25_0000
daily.2017-07-26_0000
daily.2017-07-27_0000
daily.2017-07-28_0000
daily.2017-07-29_0000
daily.2017-07-30_0000
daily.2017-07-31_0000
hourly.2017-07-30_1200
hourly.2017-07-30_1600
hourly.2017-07-30_2000
hourly.2017-07-31_0400
hourly.2017-07-31_0800
```

You can go into these folders to see what files existed at that time. Go into one of those folders. Look at the files and folders in it. If you see something you need to undelete, copy it back into the original folder (not into the `.snapshot` folder). To copy a whole folder, use `cp -r` instead of `cp` so it copies any subfolders as well.

Problem 7: What is the latest time stamp for your home folder in the `.snapshot`? Hint: use `cd` and `ls -l` to find out. (The `-l` means "long" option which prints extra information about a file.)

Make sure you have all answers to the seven problems above saved in your *lab01.txt* before moving on. And make sure your *lab01.txt* file is in your *~/csci204/* folder.

4. Python warm-up

In this part of the lab, you refresh your knowledge of Python and learn some new useful commands.

1. **Lists**

Python has a built-in list data type. A Python list is a **mutable** object with fields for length, capacity, and an array (a contiguous set of memory) of data. It allows a special quick and easy access using operators '[' and ']' (very common in programming languages). **Mutable** means it can be edited.

Try the following commands for lists and try to figure what they are doing:

```
a)
list = [1,2,3,4,5,6,7,8,9]
print(len(list))

b)
list = [2,4,6,8]
list2 = [10,12,14]
list3 = list + list2

c)
list = [1,2,3,4,5,6,7,8,9]
slice = list[2:5]

d)
list = [1,2,3]
list.append(4)

e)
list1 = [1,2,3]
list2 = [4,5,6]
list1.extend(list2)

f)
list = [1,2,3]
list.insert(2,50)

g)
list = [1,2,3]
list.pop(1)
```

In your hand-in file, answer the following questions.

Problem 8: What would be returned by the `pop()` function with no parameter?

Problem 9: In one or two lines, explain the difference between `append()` and `extend()` functions for lists.

2. **Strings**

Strings allow [ ] access similar to lists for reading but strings are **immutable**, which means they cannot be changed (we can however create a new string when we want changes).

Try the following commands for strings and figure what they are doing:

```
a)
'hi' + 'ya'
 'hi'*3
len('hihihi')


b)
word='Hello, world!'
word[0:3]

c)
 word.capitalize()
 word.lower()
 word.upper()

 d)
 word.isalpha()
 word.isdigit()

 e)
 sentence= 'hi there, she said.'
 sentence.count('hi')

 sentence.find('hi')

 sentence.split()
```

In your hand-in file, answer the following questions.

Problem 10: What would be returned by `find()` with no parameter?

Problem 11: In one or two lines, explain the difference between `isalpha()` and `isdigit()` functions for strings

3. **Dictionary**

Python has a built-in list dictionary type. Similar to a Python List that associates indices with values, a Python dictionary associates names with values. It also allows quick and easy access using [ ] (very common in programming languages). Like a Python List, it is mutable (it can be edited).

Try the following commands for lists and try to figure what they are doing.

```
a) ages = {'Alice':20, 'Bob':18, 'Cecil':19}
   print(ages)  # look at the order of names
```

```
b) favorite_colors = {}
   favorite_colors['Alice'] = 'red'
   color = favorite_colors['Alice']
   ages['Alice'] = ages['Alice'] + 1
   ages['Dan'] = 20

c) for person in ages:
      print(ages[person])
   if 'Alice' in ages:
      print('Found Alice')

d) len(ages)

e) del ages['Alice']

f) del ages
```

In your hand-in file, answer the following questions.

Problem 12: What is the difference between `del` with a key and `del` without a key (see `e)` and `f)` above)?

Extra: What happens if you delete the only key in a dictionary using the key?

Problem 13: Is person in "`for person in ages`" the keys or the values or the indices of the `ages` dictionary?

Problem 14: Is `person` in the following list code, the indices or the values of the `ages` list?

```
ages = [20, 18, 19]
for person in ages:
    print(person)
```

4. **Multi-dimensional Lists**

Python allows multi-dimensional lists. A two-dimensional list might be used to hold image data. A three-dimensional list could hold multiple images and form a movie. This can be expanded to any number of dimensions. Here is how to store the value `True` into a nine-dimensional list (you have to have such a list first).

```
nineD[3][5][2][7][1][8][3][9][4] = True
```

Try the following commands for lists and try to figure what they are doing.

```
a)twoD = []
  for a in range(4):
      twoD.append([])
      for b in range(5):
          twoD[a].append(0)
```

```
b) twoD[2][3] = 5

c) for a in twoD:
       print(a)
```

In your hand-in file, answer the following questions.

Problem 15: Does this 2-D list have 4 rows or 4 columns?

Problem 16: In `twoD[2][3]` is the 2 the column or the row?

Problem 17: In the `for` loop in `c)`, is `a` the value of an element, or a column, or a row?

5. Printing and Handing in your lab

1. **Printing**

`a2ps` is a Linux print command which prints files 2 pages to a side, tries to double side the paper, and adds your name and the date.

Print your lab file by typing *`a2ps lab01.txt`* and it will come out of a nearby printer. You either need to `cd` into the correct folder or use a shortcut path for *lab01.txt*.

2. **Submission**

Hand in a printed copy of the file *`lab01.txt`* to the TA in lab or by the deadline.