CMSC 113 – COMPUTER SCIENCE 1 (Prof. Kumar)
Lab#1: Hello Computer Science

Objective

Familiarize yourself with Linux (command line shell, basic commands), and also write your first Java program and learn how to compile and run a program.

Linux

This lab has three parts- PARTA, PARTB, and PART C. Our goal today is to learn some basic Linux commands to navigate files and directories (PART A), learn how to create and edit text files (PART B), and finally, how to write, compile, and run Java programs. Please follow the handout in the order written and DO everything that is requested of you. Do not hesitate to ask the instructor in case you have any questions during the lab. This is highly encouraged! You may not be able to complete the entire lab in today's section. This is by design. Please, take some time this week, before your next class, to return to the lab to finish. We recommend making full use of TA Hours (6:00p to 10:00p) for this.

At the end of each part, you are asked to fill out the **Lab Report** (last sheet in this handout). This charts your progress in this lab. Please remember to submit this handout to your instructor before leaving the lab. Submitting the report will count as proof of attendance in the lab. You are not required to complete everything in the Lab Report.

Part A: Working with the Linux command line

Before doing any Java programming, let's do a warmup on learning and working with the command line. First, log in to your Linux account. If you do not have a log in, please see your instructor.

Next, you will open a CLI window (*aka* Terminal emulator). From the **`Applications`** drop-down menu (see the top bar of your screen), navigate into the **`System Tools`** sub-menu and select **`MATE Terminal`** (MATE is pronounced *ma-tay*). A window will pop up in the middle of your screen and it will have a prompt that looks like this:

```
[xena@codewarrior ~]$
```

The above is a command prompt, implying that the system is ready for your commands. The command prompt is preconfigured to show your username (in this case, `xena`), the symbol `@`, followed by the name of the computer you are logged into (in this case, `codewarrior`). This is then followed by the symbol "`~`" (which stands for your current home directory), and finally ends with a "`$`".

You type commands at the prompt and when you hit the RETURN key, the command is executed or carried out. For example, enter the command "`whoami`":

```
[xena@codewarrior ~]$ whoami
xena
```

The `whoami` command reports back the username of the person currently logged in (in this case, `xena`). Next, let us learn some other basic commands.

What is my present directory (i.e. home directory): `pwd`

```
[xena@codewarrior ~]$ pwd
/home/xena
```

Directories are organized in a tree structure. Reading the result of the above command from left to right, "/" represents the root directory, home is a subdirectory of / that is the parent directory of all users on this computer, of which xena is one. After the first /, the rest of the /'s are used to separate subdirectories under them. The string /home/xena is also called a **directory path**. For users, the symbol "~" is a shorthand for their home directory /home/xena. More on this later.

We will make a new directory, called cs113 (/home/xena/cs113), so that you can store all your files related to this course in or under that directory. To make a new directory, use the command: mkdir

```
[xena@codewarrior ~]$ mkdir cs113
[xena@codewarrior ~]$
```

While there is no visible result, the prompt reappears, this creates a directory, cs113, in the home directory (xena). Since directories are organized in a tree structure cs113 is a subdirectory under your home directory. To examine the contents of a directory, the command ls is used (ls stands for show a listing of this directory):

```
[xena@codewarrior ~]$ ls
abc.txt     cs113 hello.java
letters     mail
```

It appears from above that xena's home directory contains five items: a text file-abc.txt, the cs113 directory (just created), a Java program-hello.java, etc. Thus, in Linux, files and directories coexist in all directories. One way to tell files apart from directories is to note the file extension(s). For example, ".txt" indicates a text file, ".java" is a Java program, etc. Later we will see how you can tell which is which.

You can navigate in and out of directories using the cd command (cd stands for change directory):

```
[xena@codewarrior ~]$ cd cs113
[xena@codewarrior cs113]$
```

Look at the new prompt, it clearly indicated that you are now in the cs113 directory. Go ahead and issue the pwd command now:

```
[xena@codewarrior cs113]$ pwd
/home/xena/cs113
```

Also, use the ls command to examine its contents. It should be empty. You will just get the prompt back.

The cd command can be used to navigate to any directory. You will use it to navigate up and down a directory tree. For example, when you are in the cs113 directory (as you would be if you are following along), you can go up into its parent directory (/home/xena) by doing:

```
[xena@codewarrior 246]$ cd ..
[xena@codewarrior ~]$
```

Try it one more time:

```
[xena@codewarrior ~]$ cd ..
[xena@codewarrior /home]$ pwd
/home
```

Thus, "`..`" is shorthand for the parent directory. You can also enter the entire directory path to go to that directory:

```
[xena@codewarrior ~]$ cd /home/xena/cs113
[xena@codewarrior cs113]$ pwd
/home/xena/cs113
```

No matter what directory you are in, you can always get to your home directory by just typing the `cd` command by itself:

```
[xena@codewarrior home]$ cd
[xena@codewarrior ~]$
```

Also, try the command: `cd ~`

What does it do? Next, try this: Navigate to go to the root directory (`/`). Check, using `pwd`, to make sure that you are there. Check its contents (using `ls`). Then, to go back to your `cs113` directory, enter the command: `cd ~/cs113`

Now that you are comfortable travelling in and out of directories, we can learn about copying files from one directory to another. The simplest form of a copy command is:

```
cp item1 item2
```

This command creates a copy of file `item1` into a file named `item2`, both in the same directory. Alternately, you can also specify to copy a file into another directory:

```
cp item1 directory-path
```

This command creates a copy of `item1` into the directory specified. The resulting copy will also be named `item1`. See item#5 in the exercise below for an example.

Exercise 1: Do the following:

Navigate to the directory `~dkumar/CMSC113/Lab1`

Check its contents, using `ls`.

You will notice a file named `README.txt`

In order to read the contents of the file you can use any of the following commands:

```
cat README.txt
more README.txt
less README.txt
```

These commands will each show the contents of the file specified. You will not notice any difference in the way these commands behave. We will examine these later.

Copy the file `cli.tar` into your `cs113` directory: `cp cli.tar ~/cs113/`

Go back to your `cs113` directory (`cd ~/cs113`).

Check to see if the file `cli.tar` is there. If you are lost at this point, please ask your instructor!

Issue the command: `tar xvf cli.tar`

The `tar` command expands the contents of archive files (extension `.tar`). Thus, the result of expanding `cli.tar` in the steps above will create several files and directories in your `cs113` directory.

Before starting the next exercise, please answer question 1 in Lab 1 Report.

Exercise 2: Using the commands you have learned above, answer these questions:

1. What is your home directory? (Hint: `pwd`) _____
2. Where did you download the `cli.tar` file to? _____
3. How many `.txt` files were expanded from `cli.tar`? _____
4. How many directories were expanded from `cli.tar`? _____
5. Move (`mv`) all the `.txt` files out from their directories, putting them all in the `cli` directory. The move command (`mv`) is used similar to the copy (`cp`) command, except instead of creating a copy of the file specified, it physically moves it to the specified destination. If you are having any difficulty with this step, consult your instructor.

Run `cat *.txt`. This will print out the contents of all the files. What message do you see?

_____

Before starting the next section, please fill out Question 2 on the Lab 1 Report.

PART B: Creating/Editing a text file

In order to create a text file (or a program file), you can use a text editor. Linux offers several editors. Some of the more popular choices are `vi, vim, Sublime, emacs, Atom`, etc. `vi, vim`, and `emacs` can be used to create/edit a file from within the command line. Emacs and Atom also offer a WIMP/GUI interface. For this lab, use the Atom editor. You can run it as an application by using your mouse (remember the mouse?!) to drop down the `Applications` menu, select the `Accessories` sub-menu, and selecting `Atom` (second item on that menu). The editor will start and give you a large window that fills your screen.

Exercise 3: Create a new text file.

From the `File` menu in Atom, create a new (untitled) file and enter the following text:

```
Talking Java

Though clarity & sense we seek
We're prone to misinterpretation
For limitless communication
In Java only we must speak!

      -: Martynas Petkevicius, 2013
```

Next, again using the `File` menu, save the file in your `~/cs113/Lab1` directory with the name `java.txt`. Go back to your terminal window and check to see that the file is now present in the `Lab1` directory. Check its contents (using `cat/less/more`) to see that the above text is there.

Before starting the next section, please answer Question 3 in the Lab 1 Report.

PART C: **Creating your First Java Program**

As shown in class, creating and running Java programs requires three steps:

- Use an Editor to write the program and save it in a file (extension **`.java`**) – **`Atom`**
- Compile the Java program. Correct any syntax errors reported – **`javac`**
- Run the program – **`java`**

Let's see how we do this. First, we need to have a program we'd like to run:

```
class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   } // main()
} // class HelloWorld
```

1. **Use an Editor to create a program/source file.**

   Using **Atom**, as you did in PART B, enter the program above into a file called `HelloWorld.java`. The name of the file should be the same as the name of the class (always!). Make sure you have saved the program in the `cs113` directory. [Optional: You may want to create a directory, called **`Lab1`**, and do this there.]

2. **Compile the program.**

   To compile the program, use the following command:

   ```
   [xena@codewarrior home]$ javac HelloWorld.java
   [xena@codewarrior home]$
   ```

   Depending on how correctly you typed your program, you may or may not have any syntax errors. In case there are no errors, the prompt will be returned as shown above. Otherwise, these will be reported following the command. You will then have to correct the errors in the Atom window, save the file, and then try to compile again.

   So, what is the purpose of compiling the program? Well, one is to detect and ensure that what you entered is a correct Java program. Second, to translate the Java program into Java byte code. This is essentially a version of your program translated into a more primitive language that a Java Virtual Machine (JVM) will be able to understand and run it. More on that in class.

   The byte code generated by the compiler is stored in a new file. In this case, since we defined the class `HelloWorld`, the file will be called `HelloWorld.class`. Go ahead and look at the contents of your `cs113` directory (using `ls`). You will see the file `HelloWorld.class` sitting there. We are now ready to run your program.

3. Run the program

You run the program by invoking the JVM (which is called `java`). The JVM needs to know the name of the class that makes up your program (i.e `HelloWorld`). Here is the command:

```
[xena@codewarrior home]$ java HelloWorld
Hello, World!
[xena@codewarrior home]$
```

The program runs, you can see its output on the line after the java command. And a new prompt is returned. You can now run the program again, using the same command.

Exercise 3: Write a new program, called `JavaJoe`, that prints out the following lyrics:

```
I love coffee
I love tea
I love the Java Joe
And it loves me
```

Exercise 4: Write the program, `UseArgument` that is described on Page 7 (Program 1.1.2) of your text. It is shown below:

```
public class UseArgument {
   public static void main(String[] args) {
      System.out.print("Hi, ");
      System.out.println(args[0]);
      System.out.println(". How are you?");
   } // main()
} // class UseArgument
```

You will store it in a file, `UseArgument.java`. Compile it (using `javac`), and run it using the command:


```
java UseArgument <your name>
```

Time to Digest

In part A, you learned some basics of using the Linux command line interface (CLI) through a terminal window. Review the following commands:

whoami
pwd
mkdir
ls
cd
cp
mv
cat/more/less
ETC.

In PART B you learned how to create basic text files. And, as you saw in PART C, all program files are also text files. In PART C you learned how to create, compile, and run simple Java programs.

This is a good start. Before next lab, please review Section 1.1 of your text and try out all Exercises (1.1.1 through 1.1.6).