CMSC246 Systems Programming - Lab 1

This lab has three parts- PARTA, PARTB, and PART C. The 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 C 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!

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.

Part A: Working with the Linux command line

Before we get to C 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 `Konsole` 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, some other basic commands.

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

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

Remember from class that 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.

Make a new directory, called `CS246` (`/home/xena/CS246`), 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 CS246
[xena@codewarrior ~]$
```

While there is no visible result, the prompt reappears, this creates a directory, `CS246` in the home directory (`xena`). Directories are organized in a tree structure. Thus, `CS246` 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   CS246     hello.java
letters   mail
```

It appears from above that `xena`'s home directory contains five items: a text file-`abc.txt`, the `CS246` 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. Beware, extensions are not enforced in any way in Linux.

To get more information about the contents of a directory ls the command `ls -la`. Examine the output of this command to find how you can recognize subdirectories for other files.

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

```
[xena@codewarrior ~] cd CS246
[xena@codewarrior CS246]
```

Look at the new prompt, it clearly indicated that you are now in the `cs246` directory. Go ahead and issue the `pwd` command now:

```
[xena@codewarrior CS246] pwd
/home/xena/CS246
```

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

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 `CS246` directory (as you would be if you are following along), you can go up into its parent directory (`/home/xena`) by doing:

```
[xena@codewarrior CS246] 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/CS246
[xena@codewarrior CS246] pwd
/home/xena/CS246
```

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 ditrectory (`/`). Check, using `pwd`, to make sure that you are there. Check its contents (using `ls`). Then to go back to your `CS246` directory, enter the command: `cd ~/CS246`

Next, copying.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:

1. Navigate to the directory `~gtowell/Public246/Lab1`
2. Check its contents, using `ls`.
3. You will notice a file named `README.txt`
4. 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.

5. Copy the file `cli.tar` into your `CS246` directory: `cp cli.tar ~/CS246/`
6. Go back to your `CS246` directory (`cd ~/CS246`).
7. Check to see if the file `cli.tar` is there.
8. 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 `CS246` directory.

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

Finally, cleanup

1. Still in the `CS246` directory issue the command below. Do this with caution. `rm` removes file and, as below, can remove directories. `rm` is permanent and unrecoverable.

```
rm -r cli
```

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

PART B: Creating/Editing a text file

First make a new directory to hold your work by doing the following

```
cd ~/CS246
mkdir Lab1
```

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`, `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, you may use any editor, but the directions below are for emacs_.

You can run emacs as an application by using your mouse to drop down the `Applications` menu, select the `Programming` sub-menu, and selecting `Emacs` (sixth 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 Emacs, create a new file (Select "Visit New File" the give the file the name `sea.txt`. Specify that your file should be put in the `CS246/Lab1` directory:

```
She sells
Sea shells
By the seashore
In Seychelles
```

Next, again using the `File` menu,. 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.

Edit the file starting from the command line

```
emacs sea.txt &
```

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

PART C: C!

Today's lab will ask you to shake the rust off your programming skills by writing several small programs in C. Try to complete as many of the programs below as you can. It is not important that you finish all of them in lab today. But make sure that you do complete all before the start of the next lab.

Hint: Remember that we use == to test for equality – never =!

I. Write a "Hello, world!" program in a new file `hello.c` that simply prints "Hello, world!" using `printf`. Compile your program with `gcc -o hello hello.c` and run it with `./hello`.

II. Write a program in a new file `sea.c` that prints the text you entered in the file sea.txt above using `printf`.

III. Write a program (in `age.c`) that asks the user for their age and tells them what age they will turn on their next birthday. Compile with `gcc -o age age.c` and run with `./age`.

IV. Write a program (in `max2.c`) that asks the user for two numbers and tells them which one is bigger.

V. Write a program (in `max.c`) that uses a loop to ask the user for many numbers, stopping only when the user enters 0. At that point, your program reports the maximum number that the user entered.

VI. Write a program (in `count.c`) that asks the user for two numbers. It then prints out all the numbers starting from the first and going to the second. It should not crash if the second number is less than or equal to the first. (You can design the behavior in this case.)

VII. Write a program (in `count_not_5.c`) that is like the previous program but skips counting any number that is a multiple of 5. C uses `%` to compute modulus, like Java and Python. The modulus of two numbers is the remainder when the first is divided by the second. So, `15 % 4` is `3`. If you want to know whether `a` is divisible by `b`, check if `a % b == 0`.

VIII. Write a program (in `fizzbuzz.c`) that plays FizzBuzz: Your program asks the user for two numbers, and it starts counting at the first and counts up to the second, one line at a time. Instead of any number that is divisible by 3, print `Fizz`. Instead of any number that is divisible by 5, print `Buzz`. Instead of any number that is divisible by 3 *and* 5, print `FizzBuzz`.

IX. Write a program (in `prime.c`) that asks the user for a number and then reports whether that number is prime. An easy (but not terribly efficient) way to check for primality of `n` is to check all numbers `m` such that `2 <= m < n` to see if `m` evenly divides `n`. If no such `m` divides into `n`, then `n` is prime.

X. Write a program (in `summult3-5.c`) that solves the following problem:

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.

XI. Write a program (in `sumevenfib.c`) that solves the following:

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.