EE 1301
Lab 0

Introduction to UNIX and C++

Welcome to the first lab in EE 1301. In this lab you will explore some computational resources needed during this course, and write a few short programs. The programming environment used in the labs runs under a variant of the UNIX operating system called Linux. This lab exercise will familiarize you with the process for creating, compiling, and running C++ programs in the Linux environment.

Register your account

Prior to lab you should have registered your **CSE Labs account**. If you have not yet done so, visit http://www.cselabs.umn.edu/ , click the "Forms" tab, then click on the link for "CSE Account Authorization Form" and register your account using the **account creation form**. Once you register your account it will take a while to be activated, so you will need to work with a partner who already has an active CSE Labs account.

Logging in

To log in from a Linux machine in one of the CSE computer labs (such as KH 1-200), pick an available machine and type your *username* and *password* in the login box. If you are logging in remotely from a Windows or Mac machine (such as your personal laptop or the machines in Keller 4-138), first go to https://vole.cse.umn.edu/, then follow the instructions to log in to a remote Linux machine.

Lab Overview

Because this is your first lab, a UNIX tutorial is provided describing UNIX commands that you will need to know. This document also describes how to create, compile, and run a simple C++ program in the UNIX environment.

Brief UNIX Introduction (adapted from C. Swanson, "Unix Tutorial")

The UNIX operating system dates back to the early 1970's and remains one of the most efficient, robust and reliable pieces of software in use today. The "operating system" refers to the collection of programs that reside on your computer and provide an interface to the actual hardware of the machine. This involves maintaining and organizing data on external devices (e.g., a hard drive), loading and running application programs, etc. You are probably familiar with graphical interfaces to operating systems such as Microsoft Windows, Apple Mac OS and others (*Note*: Mac OS is built on top of the UNIX operating system and uses UNIX for its core functions). Our lab computers employ Linux, which is a version of UNIX, so you will need to learn a few essential UNIX commands. You are encouraged to explore and learn more on your own! Later, Google "UNIX Tutorial" and go through one of the tutorials you find to get more experience with UNIX/Linux.

The UNIX Terminal
Unlike many other computer systems you might have used in the past, you will be interacting with UNIX mainly using a ***command line interface***. This means you will type commands into a terminal window, along with required and optional ***arguments***. Arguments are additional pieces of information provided to a command. For example, if you want to delete a file, you need to specify both the UNIX remove command (`rm`), and the name of the file to be deleted (removed). The following remove command with argument `file.txt` would delete the file named file.txt.

```
rm file.txt
```

In the UNIX operating system, commands are typed into a **terminal** window. Open a terminal window by clicking on the terminal icon (generally located on the upper-left side of your display – it looks like a little computer monitor). Start the terminal now. Seek help from your TA if you can't find the terminal application.

The Command Line Interface
The operating system will prompt you for a command, using a short text string that looks something like the following.

```
username@machine%
```

where *username* is your UNIX user name and *machine* is the machine you are currently logged in to. The `'%'` symbol is there to let you know that the system is ready for another command. Sometimes, the `'$'` symbol is used instead of the `'%'` symbol. Commands can be edited while they are still being typed using the left and right arrow keys or the backspace key. ***Important note***: UNIX commands are ***case-sensitive***. Case-sensitive means that upper- and lower-case letters are treated differently, for example, the remove command is `rm`, not `RM` or `Rm`.

Most UNIX commands are shortened names that describe what they do. For example, `cp` stands for *copy* and `mv` stands for *move*. If you ever forget what a particular command does or how it is used, you can use the **`man`** command to learn more; `man` stands for *manual.* The manual page (man page) for a command will tell you everything you need to know (and more) about a particular UNIX command. Simply type: `man` *`command-name`* `[ENTER]` to display the manual page. Press [SPACE] to scroll through the pages of information and 'q' to quit out of the manual. You can even use the `man` command to learn about `man` itself (e.g., `man man`)! Try learning more about a command you already know, like `rm`:

```
man rm[ENTER]
```

Files and Directories
You are probably familiar with the concepts of files and folders. A **file** is a collection of data that has a specific name associated with it. A **folder** is a special file that contains zero or more files or other folders. In UNIX, folders are called ***directories***, but they are essentially the same thing. Files and directories allow users to store and organize their data.

File and directory names are **case-sensitive**, and special care should be taken when naming them. Avoid using spaces or special characters, except for the underscore, hyphen, and period. Certain special characters are forbidden in filenames, but others are allowed, and some may cause problems. Also, avoid giving files the same name as UNIX commands.

Navigating the File System
All of your files and directories are contained within your ***home* directory**. When you first open a new terminal, your *home* directory will be your active or *working* directory. The working directory is simply the directory you are currently viewing or working in. You can use the *print working directory* command, **`pwd,`** at any time to show you what your current working directory is:

```
pwd [ENTER]
```

You should see something like:

```
/home/username
```

This is the complete *path* from the first (root) directory to your current working directory. Each directory name in the path is separated by forward slash characters.

To list the names of all of the files and directories in your working directory, type the *list* command, **`ls`**:

```
ls [ENTER]
```

All file names will be listed; directories are sometimes displayed with an ending forward slash. Directories can be created and deleted using `mkdir` (*make directory*) and `rmdir` (*remove directory*). Try making a new (sub)directory called "EE1301".

```
mkdir EE1301 [ENTER]
ls
```

You can change to a different working directory with the *change directory* command, **`cd`**. To enter your newly created `EE1301` directory, use the following command.

```
cd EE1301 [ENTER]
```

Now we will introduce some special names for directories. The double-dot (`..`) stands for the ***parent* directory**, which is the directory above the current directory that contains the current working directory. We can use this to "move up" one level in the directory hierarchy. Since your home directory is the parent directory of the EE1301 directory, moving up one level brings you back to your home directory. Try it!

```
cd .. [ENTER]
pwd    [ENTER]
```

The tilde (~) is shorthand for your *home* directory. You can return to the home directory (from anywhere) by typing:

```
cd ~ [ENTER]
```

Now that we have returned to the home directory, we can remove our EE1301 directory. Type

```
rmdir EE1301 [ENTER]
ls [ENTER]
```

Note that `rmdir` will only remove empty directories (i.e., directories that contain no other files or directories). If you want to remove a directory, you must empty it first.

Creating C++ source code files

To write a C++ program, you normally begin by using a *text-editor* to create a file with a `.cpp` extension, e.g., `test.cpp`. A text editor is a relatively simple program (application) that works like a stripped-down word processor. It allows you to enter/modify text and save the text to a file. There are a number of text editors available. Unless you already have a favorite editor, use the `geany` text editor for this lab assignment.

The file you create contains C++ statements and is known as a ***source code*** file. Source code is written in a format that humans are able to read and understand, but the computer can't read and execute source code. C++ is a ***compiled*** computer language. The C++ code you write cannot be directly interpreted by the computer. In order for the computer to execute a program written in C++, the program must be compiled into instructions that the computer can understand. The process of transforming the C++ code you write into instructions that the computer can read is called **compilation**. There are many C++ compilers, but the one used for this class is named **g++**. g++ takes source code files as input and creates a file called a binary that a computer can execute.

If the compiler does not understand what you wrote in your source code, it will complain by reporting one or more ***syntax* errors**. If your code contains syntax errors, you need to look at the error messages, identify the errors, and then use your editor (e.g., geany) to fix any errors in your source code. Sometimes your program will compile correctly but will fail to execute to completion due to a ***run-time* error**. For example, attempting to divide a number by zero causes a run-time error because the computer does not know how to store the result of the division operation (infinity). Finally, your program may compile, and even execute, but produce incorrect results due to one or more ***logic* errors**. If your program contains logic errors, you will need to go back to the editor and fix the errors. Any time you change your source code (e.g., to fix an error), you need to **save and recompile** the program before testing it again.

First Program

Let's write our first program! It will be a short program that simply displays a message to the terminal.

First, create a special directory for this lab by entering the following commands into the terminal.

```
mkdir EE1301  [ENTER]
cd EE1301     [ENTER]
```

The first command creates a directory named `EE1301` (Note: keeping different labs and assignments in different directories is a good idea to help keep your files organized). The second command moves you into the directory you just created.

Creating the source code file

Start the geany editor and create a file called `lab0.cpp` by entering the following command.

```
geany lab0.cpp  [ENTER]
```

Once geany is running, you can enter source code into the editor. Figure out how to copy and paste the following program from this document to the geany window. Ask a TA if you have trouble. Note that computer languages like C++ are very precise, and the compiler has no tolerance for errors! Every symbol in this program has significance and must be copied *exactly* as it appears. Note the use of semicolons in some of the lines.

```
// Lab 0
// Replace this text with your name
// Type today's date here
//
// This program displays a message to the screen
#include <iostream>
using namespace std;

int main()
{
   cout << "Hello!" << endl;
   return 0;
}
```

Next, edit the file by replacing the text on the second and third lines with your name and today's date, respectively. Once again, ask a TA if you are unable to figure out how to do this. Note that simply **typing or pasting text in the editor program does not actually *create* the file.** You must explicitly ***save*** the text to the file. To save your changes, select "File" and "Save" in the pull-down menu. We'll use this source code later, but for now it will provide an opportunity to learn some additional UNIX commands. Once you have finished typing in the program, save it and then close (quit) geany.

A little more UNIX

Before you compile and run the program, let's learn a few more UNIX commands. Your `lab0.cpp` source code is now saved as a file within the `EE1301` directory. Suppose we really had intended to put the file in a directory called `lab0` under our home directory. Making this change will give us an opportunity to practice some UNIX commands. If you aren't already there, change to your home directory.

```
cd ~ [ENTER]
```

Now make a new directory named `lab0`:

```
mkdir lab0 [ENTER]
```

then change your working directory to the `EE1301` directory:

```
cd EE1301 [ENTER]
```

We'll use the *copy* command, `cp`, to copy files from one directory to another. The copy command takes the form

`cp` *source-file destination-file*

where *source-file* is the name of the file to be copied, and *destination-file* is the location to which we want to copy the source file. Use the list command to list all the files in the `EE1301` directory:

```
ls [ENTER]
```

Now, copy the file to the **`lab0`** directory using the `cp` command.

```
cp lab0.cpp  ../lab0/ [ENTER]
```

The `../lab0/` argument tells the `cp` command to find the directory named `lab0` in the *parent* directory, and copy the file there. Now use the *list* command to list the names of the files in the `lab0` directory, to make sure they were copied successfully.

```
    ls ../lab0/  [ENTER]
```

Once you have verified that you copied the files correctly, you can delete the files in the `EE1301` directory using the *remove file* command.

```
    rm lab0.cpp  [ENTER]
```

Be careful! Unlike other computer systems, there is no "undelete" or "Recycle Bin" in UNIX. Once you delete a file, it's gone for good in most cases. Now return to your home directory.

```
    cd ~ [ENTER]
```

Copying files from `EE1301` into the `lab0` directory is good practice. However, we could have saved ourselves some work by simply renaming the `EE1301` directory! Try it yourself. Use the *move* command to rename a directory:

```
    mv EE1301  hello [ENTER]
    ls [ENTER]
```

Change it back using:

```
    mv hello  EE1301 [ENTER]
    ls [ENTER]
```

Note that the `mv` command can be used to rename either a file or a directory.

Recall that if we want to delete a *directory* instead of a file we must use the *remove-directory* command: `rmdir`. Since our source code file (lab0.cpp) is now in the `lab0` directory, we no longer need the `EE1301` directory, so let's delete it.

```
    rmdir EE1301  [ENTER]
    ls  [ENTER]
```

The `EE1301` directory should be gone from your home directory.

Lastly, we will discuss two more convenient functions provided by the command line. These are ***command history*** and ***tab completion***.

Tip: Command History (VERY USEFUL, USE THE ARROW KEYS)
You may have noticed already that pushing the up arrow in a terminal window brings up previous commands that you have entered. You can use this to your advantage to recall previous commands you have typed in that terminal window. This is especially useful if you make a mistake in a long command. You can simply press the up arrow until you arrive at that command, move the cursor to the location of the error using the left and right arrow keys, delete the error, and type in your correction. Try it with the following command.

```
    makdir hello [ENTER]
```

This should give an error. Press the up arrow once to recall the command. Then press the left arrow until the cursor is on the 'k'. Now press backspace to remove the erroneous 'a'. Finally, press [ENTER] to retry the command. **Note:** The cursor does not need to be at the end of the command to enter the command. The terminal will take all text written in the line as the command, regardless of cursor position. I.e., pressing ENTER while the cursor is still on the ‘k’ will enter the full command.

Tip: Tab Completion (ALSO VERY USEFUL)
Often, you will have long filenames that are difficult to type. Tab completion will save you typing. You can simply type the first few letters of a file or directory name, press the tab key, and UNIX will try to complete the name for you. Try this:

```
cd ~ [ENTER]
cd h [TAB]
```

The terminal should "fill in" the rest of hello. But what happens when multiple files start with the same sequence of characters? Try this:

```
mkdir hydrogen [ENTER]
cd h [TAB, TAB]
```

The terminal should display the names of all files and directories that start with ‘h’. In this case, hello and hydrogen both qualify. Type

```
y [TAB]
```

and the terminal should fill in the rest of hydrogen, since it is the only file that begins with “hy”.

You've now learned to use several basic (and important) UNIX commands for manipulating files. If you are still unsure of what you are doing, discuss it with your TA before continuing. You may also find it useful to explore the wealth of information and tutorials available on the Internet. Simply search for “unix tutorial”.

Check
To check your knowledge of UNIX, make a list of six to ten UNIX commands. Try to list as many as possible before looking up the commands in this lab write-up. Also make sure you know what the commands do (use the `man` command if you are unsure).

Compiling C++ programs

As described earlier, the process of creating a C++ program involves creating a *source-code* file using an editor of some sort, then transforming the source-code file into an executable program (containing instructions that the computer can read and execute) using the C++ compiler (`g++`). The compiler will create an executable file that the computer can run.

To compile the `lab0.cpp` program you created earlier, make sure you are in the `lab0` directory and then type the following on the UNIX command line to run the **g++** compiler on `lab0.cpp`.

```
g++ lab0.cpp -o lab0   [ENTER]
```

If you entered the `lab0.cpp` source code into the editor correctly, g++ should compile it without any errors and produce an executable program named `lab0`. The **`-o`** option specified on the command line indicates that the next argument you type (**`lab0`** in this case) is the name of the executable.

Use the `ls` command to make sure both `lab0.cpp` and `lab0` (the executable) are in the directory listing.

```
ls   [ENTER]
```

If the compiler produced one or more error messages, then you probably made a typing or copying error in the source code. Check your program again and make sure it is *identical* to the one in the lab write-up. If you find a place where the two differ, run geany again (`geany lab0.cpp`), correct your program, re-save it, and recompile it (`g++ lab0.cpp -o lab0`).

Executing your program
Once the program compiles, run it by typing the executable file name preceded by `./`

```
./lab0
```

This should display **Hello!** to the screen. What happens if you remove the "<< endl" part in the source code? Once your program is running correctly, show the following to the TA.

A. A listing of your program using the `cat` command (type `cat lab0.cpp`)
B. The compilation of your program using the `g++` command
C. A test run of your program.

Logging Out (DO NOT TURN OFF THE COMPUTER)

If you are using a Linux machine in one of the CSE labs, log out of your account using one of the following commands.

```
logout   [ENTER]
```

or,

```
exit   [ENTER]
```

All of the windows you were using should disappear and the login screen should reappear.

If you are using vole, you can choose to log out or close the browser window.

Remember to logout at the end of each lab.