CS 305 Lab 1: Introduction to C, Linux, and compiling
Cenek, Fall 2018

This lab introduces you to some of the basics of using the Unix/Linux operating system and the emacs text editor. You will also learn how to create, compile and run a C program in Unix.

This lab has a total of 100 possible points (30 points from pre-lab and 70 points from lab). If you do not finish the checkpoints during the lab session, write your answers on this sheet and include any screenshots/printouts that show your work. It is due at the next lab session.

Objectives

Upon completion of the laboratory exercises, you will be able to do the following:

- Login/logout to/from a Linux computer.
- Look up and use Unix commands.
- Create, edit and save a text file using emacs.
- Create, compile and run a C program using gcc.

You should work with a partner throughout this lab. Person 1 is the partner whose first name comes first alphabetically and Person 2 is the partner whose first name comes second alphabetically. If you have taken or are currently taking CS 376 you will find that some of this lab is review. Please be patient with your lab partner if s/he is not taking CS 376.

Part 1: Logging in (person 1 should drive)

Because the Linux computer is remotely accessed, the login and logout process requires two steps. First, you log into the PC and then login to the Linux machine.

1. Login to Windows using person 1's campus username and password.

2. Open Mobaxterm. Click on the start local terminal. Person 1 will log into egr3.up.edu for the labs in CS 305.

username = Your UP login

In the center black window, type:

```
ssh -l username egr3.up.edu
```

This will connect you to a Linux machine.

3. At the prompt, type your UP password. Say "no" for remembering the password. You may see a message welcoming you to the Linux machine or a message about the last time you logged into egr3.up.edu. You may be asked to add egr3.up.edu to a list of hosts. If so, type yes.

4. Exit the Linux machine by typing at the prompt:

```
exit
```

5. Person 2 should log in, following the above 3 steps.
```
ssh -l username egr3.up.edu
```

If egr3.up.edu is not working for some reason this semester, you may use egr2.up.edu as an alternate. Note that your files are separate on both machines, so you will need to set your .bashrc file for both machines (see instructions below).

Part 2: Using man (manual) command to get help (person one should drive)

1. The *man* command is used to view the Linux online documentation. Read the following information about the `man` command.

Note: when you give a man command, and there is too much information to fit on one screen, it gives you one screenfull and pauses until you hit the space bar; once you do so, it gives you the next page. If you want to quit (e.g., before reaching the end of the file), you can type 'q'. This behavior is actually implemented using the 'more' command; you can find out more details by typing man more.
Type:
```
man more
```

If you don't know the name of a command but you know a keyword that might be part of the command's description, use the -k switch. Thus, if you wanted to know about commands that are associated with printing, you might say man -k print or (if you don't want things scrolling off the screen) man -k print | more:
```
man –k print
```

This same command (man -k) is also called apropos; the two are identical. Use whichever version you find easier to remember.

Man is also used to tell programmers about system functions and data formats that can only be accessed in a meaningful way by writing a program. If you only want to find out about command-line commands you can say:
```
man –S 1 tail
```

This is because virtually all of the command-line commands are in Section 1 of the online manual.

When you use the '-k' switch, you can tell which are command-line commands by looking at the command names with a '(1)' after them.

2. If you ever want additional terminal windows when logged into a Linux machine, you can open an additional window by typing:
```
xterm &
```

This will be another window where you can type commands, as you did in the main screen. In the new window, type:
```
man head
```

3. Close the xterm window and use the main (black) window in mobaxterm.

Part 3: Edit text with emacs (person two should drive)

1. The emacs program is a standard text editor under Unix/Linux. There are two versions:
   - A GUI (graphical user interface) version, which allows both *keyboard* and *mouse* interaction.
   - A keyboard-only version which, for example, is appropriate for running over the internet when an X-Windows system is not available. This is emacs –nw.

Most students will use the GUI version, so skip down to that part. The text-based version documentation is below in case you want to reference it.

Emacs -nw
You can run the text-based version of by typing emacs -nw (or emacs -nw *fileName*) to the command-line prompt. Try this now.

```
emacs –nw
```

You should see a text version of emacs, with “Welcome to GNU Emacs, one component of the GNU/Linux operating system.” at the top. To close emacs, type Ctrl-x and then Ctrl-c.

The following is a list of basic emacs commands. In the list below, a character prefixed by a 'C-' means that the control key should be held down as the key is being pressed. Thus, 'C-p' (pronounced "control p") means that the control key should be held down as that 'p' key is being pressed. The 'M-' prefix means that the escape key should be pressed before the key is pressed. Thus 'M-f' (pronounced "meta f") means that the "escape" key should be pressed, followed by the 'f' key.

- Files.
  - C-x C-c exit emacs
  - C-g abort current command
  - C-x C-f open file
  - C-x C-s save file
- Cursor movement. (Note: when using emacs , you can also use the arrow-keys to move around.)
  - C-f move forward one character
  - C-b move backward one character
  - C-a move to beginning of current line
  - C-e move to end of current line
  - C-n move down one line
  - C-p move up one line
- Cut/paste.
  - C-d delete next character
  - <delete> delete previous character
  - C-k "kill" rest of current line (or delete 'newline' if at end)
  - C-y paste most recently killed text at the current location
- Screen/windows. (Note: when using an X-windows version, you can also use the scrollbar to scroll.)
  - C-l redraw (possibly garbled) screen

    - C-v scroll up
    - M-v scroll down
    - C-x 1 delete all windows but current one
    - C-x 2 split window
    - C-x o go to next window
- Search/replace.
    - C-s search for a string (forward)
    - C-r search for a string backward
    - M-% interactive string replacement

Emacs (GUI version)
You can run a graphical text editor by typing `emacs &` (or `emacs fileName &`) at the command line prompt (assuming X-windows is working). Try this now by typing `emacs &`.

This editor is much more user friendly and allows you to manipulate the text (highlight, drag and drop, etc.) using your mouse, which is probably more convenient for you. Also, you can manage files, cut and paste, spell, and undo commands. There are many other commands but we will not review them at this time.

You will find that this editor is very similar to ones that you have already used such as Microsoft's WordPad or Notepad. Unfortunately, there is a minor incompatibility between Unix- and Windows-based text files in that end-of-line is represented differently. WordPad seems to handle Unix-created text files just fine, but some versions of Notepad do not; similarly, some text files show a '^M' at the end of each line when editing a Windows-created file.

2. Use the File->Open File button. Edit the file in your Linux account named `.bashrc` (yes, it starts with a period). You will need to click the checkbox for hidden files (find the file and then hit enter). Add the following three lines to the end of the file.

```
export PATH=$PATH:.
umask 0077
(a blank line)
```

3. After you have finished editing the file, save it by hitting C-x C-s or using File->Save. Type the following command at the $ prompt in the Mobaxterm main window:
`source .bashrc`

This will configure your account so the default file protections on the operating system are more appropriate and include the current directory in the path.

The other partner should log into egr3.up.edu and update their `.bashrc` file and source it.

Part 4: Files, directories, and navigation (person one should drive)

1. From your home (where you logged in) directory (i.e., folder), type the following commands after the $ prompt to create a new directory called lab1, and move into it by doing the following:

```
mkdir lab1
cd lab1
```

2. List the contents of your directory. You should see no files as contents of this newly created directory.

```
ls
```

3. The directory structure is a tree. If you ever get lost, you can print your working directory. Do that now:

```
pwd
```

It should say something like:

```
/home/username/lab1
```

If you ever need to up one level in the directory structure, you can type:

```
cd ..
```

If you want to go back to your home directory, you can type:

```
cd
```

3. Go to your home directory now.

4. While in your home directory, create a directory called cs305 (unless it has already been created); then move your lab1 directory into your cs305 directory; finally, move into your cs305/lab1 directory.

```
mkdir cs305
mv lab1 cs305
cd cs305
cd lab1
```

Note that the above example does not put the lab1 directory in the top-level home directory (e.g., ~username), but instead within the cs305 subdirectory. You may wish to continue this practice to keep your labs organized.

5. If you ever want to remove a directory that is empty, you can use:

```
rmdir lab2
```

BE VERY CAREFUL WHEN REMOVING FILES AND DIRECTORIES. There is no "undo" after a file or directory is deleted.

If you ever want to delete a file, you can use:

```
rm filename
```

If you need to remove all files within the current directly, you can use:

```
rm *
```

AGAIN, USING rm CAN BE DANGEROUS. For CS 305, you will likely not need to remove several files at once, so I do not recommend using rm *.

6. On the left side of Mobaxterm, you should see a directory listing. If your newly created folder cs305 does not appear, click on the move up one level icon and then click back on your home directory. You should see the folder cs305. Click on it. You should see lab1. Click on it. Now, download the file lab1.txt from Moodle to your PC (P drive might be handy). Right-click in the left window of Mobaxterm and click "Upload to Current Folder". Navigate to the file and select Open. (The purple arrow icon is also the upload to current folder tool.)

Open the file `lab1_secretText` in emacs or do a less on the file.
What is the special word of the day, located in the file? _____

If, at any time, you want to re-do a command you have already typed, you can use the up arrow to retrieve a previous command. This will save you time typing .

If you want the command line to auto complete your syntax, press tab. If the syntax is non-deteministic, it will give you the options available to auto complete the command.

Unix cheat sheet:

cd xxx	change directory to xxx
cd	change to home directory
rmdir	delete directory (must not have any other directories or files within it)
rm	delete file
ls	list contents in directory
cp	copy file/directory
mv	more file/directory
mkdir	make a new directory
pwd	print working directory

Part 5: Compiling and Running C Programs (person two should drive)

In our Linux environment, we use the `gcc` command to compile a C program. For example, to compile `my_prog.c`, we could say:

```
gcc -g my_prog.c
```

The -g indicates that debugging information should be produced. The debugger information will be useful when we get the point of using the debugger in the course. This creates an executable file with the name a.out. This is the default executable file name for a C program in Linux.

To run the program, you simply type at the command prompt the name of the executable file, `a.out`, followed by the return (or enter) key. When you run the program, it will begin executing the main function which should be declared as:

```
int main(int argc, char *argv[]) { ... }
      (or)
int main(void) { ... }
```

1. Type the command to compile my_prog.c: `gcc -g my_prog.c`

What message did you get? _____

Why did you get that message? _____

2. In your lab1 folder, use emacs to create a C program for printing “Hello, World!”. The program should be named `hello.c`.

```
emacs hello.c &
```

Type the following code into the file:

```
#include <stdio.h>

/* main program prints Hello, world */
int main (int argc, char* argv[]) {
     printf(“Hello, world!\n”);
     return 0;
}
```

3. Compile and run the program in the command-line window. (Make sure you save the file before you compile.)

```
gcc -g hello.c
a.out
```

4. Switch drivers, so the other partner gets a chance to modify and run a C program. Modify the program so that it has your two names in comments at the top of the file. Modify the program so that it prints “I love C” 50 times (each on different lines) after printing “Hello, world!” once. A for loop in C is just like a for loop in Java, except you must declare the iteration variable prior to the loop. Here is an example loop that iterates 10 times:

```
int i;
for(i = 0; i < 10; i++) {
     /* body */
}
```

6. Save the file, compile it, and run your modified program. Note: Each time you make a change to the program, you must re-compile it for the changes to take effect.

Checkpoint 1 [25 points]: Let your lab instructor/assistant see the modified "Hello world" running and your answers to the questions up to this point in the lab.

Part 6: Command line arguments (person one should drive)

In C programs, the main function has two parameters: argc and argv. The first, argc, is the number of command line arguments that come after the program executable name. The second, argv, is the array containing those command-line arguments as strings. In argv, argv[0] is the name of the program, argv[1] would be the next item,

and so forth. For example, if there is a program called my_prog, then the execution of:

```
my_prog tammy sally fred
```

would mean that argv[0] = “my_prog”, argv[1] = “tammy”, argv[2] = “sally” and argv[3]=”fred” and argc = 4.

1. Copy `hello.c` to `command_line.c`:

```
cp hello.c command_line.c
```

Open command_line.c in emacs.

Instead of always printing “I love C” 50 times, update the code to do the following:
If `argc` is not equal to 2:
    Print “I love C!” followed by a newline. (print just once)
Else:
    Print the argument after the name of the program given on the command line followed by a newline (print just once)

```
if(argc != 2) {
  printf("I love C!\n");
} else {
  printf("%s\n", argv[1]);
}
```

Note that the syntax for conditions (if and if/else) in C is the same as Java. Now that we have more than one program in the directory, it makes more sense to name the programs when we compile them instead of using the default a.out. When you compile `command_line.c`, now use the form:

```
gcc -g command_line.c -o command_line
```

2. Run the code with the following calls:

```
command_line
command_line apple
command_line apple banana
```

Checkpoint 2 [20 points]: Let your lab instructor/assistant see the output of the program executions.

Part 7: Variable types and sizes (person two should drive)

Let’s experiment with variables, their types, and the amount of space they take in memory. Download the file `sizes.c` from Moodle and put it in your lab1 folder on the Linux machine.

1. Open the file in emacs and read through the code. This program assigns different types of variables, pointers to variables, prints the sizes, and prints the values.

2. Compile and run the code. Name the executable `sizes`.

How many bytes are used to store ints? _____

How many bytes are used to store chars? _____

How many bytes are used to store doubles? _____

How many bytes are used to store pointers? _____

3. Re-run the code a second time.

Did the values for the int, char, and double variables change? Yes No

Did the values for the pointers change? Yes No

Why would the values for the pointers change? _____

4. Modify the code to add one new type of variable throughout the program. C supports several types, such as short, long, unsigned int.

What is the new type you added? _____

How many bytes are used to store this type? _____

NOTE THAT NOT ALL C PROGAMS HAVE THE SAME SIZES FOR TYPES. In Java, across all platforms, the size of an integer is always the same (4 bytes). In C, it is likely 4 bytes, but it could be defined for a certain architecture to be 8 bytes instead. The spec states that chars are at least 1 byte, ints at least 4 bytes. Also, the sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long).

C also has types that include signed and unsigned. If the type is int (without unsigned in front), it is defaulted to be a signed int. An unsigned int only stores non-negative values. Feel free to experiment with creating signed, unsigned, and regular types to see what data they can store.

Checkpoint 3 [20 points]: Let your lab instructor/assistant see the output of the modified program and your answers above.

Part 8: Printing (person one should drive)

Although Linux has commands for printing, the machines are not connected to any student-accessible printers. To print a file, therefore, we need to transfer the file back to the PC for printing. Make sure all emacs windows are closed.

Moving files back to the PC is also helpful if you want to save your lab files on your P: drive instead of the Linux machine.

Mobaxterm has built-in sftp. Enlarge the leftmost tab that says sftp on the left of your Mobaxterm app if it is not visible. You can drag the right-edge of this window to

the right to increase the width. This window shows your file directory on the Linux machine. Navigate to your hello.c file by clicking on folders until you find your hello.c file. You'll also see a hello.c~ file. Emacs saves a prior version of your file with the tilde in case you want to go back to previously saved version. Be sure to select the hello.c version.

Right-click and select Download. Navigate to the appropriate folder on the PC. Click OK.

Then, open the file in that folder (in wordpad, emacs, or any other text editor on the PC) and print it to the lab printer.

Checkpoint 4 [5 points]: Hand in the printed version of your modified "Hello world" program to your instructor. Be sure your names are in comments on this file.

Logout of the Linux machine:

```
exit
```

5. Close any windows that are still active. Close mobaxterm. Finally, log out of the PC. You are finished.

Any lab checkpoints that are not completed during the lab session are due at the beginning of the next lab session.

Appendix A: Unix commands on Mac or PC (for your reference)

The Unix commands you have learned in this lab can be applied directly in xterm windows on a Mac or Linux computer. OS X is Unix-based.

The Unix commands you have learned in this lab can be applied directly in the mobaxterm application on Windows. Cygwin is a Linux-like environment for Windows that is free for download. If you have a PC (Windows) and you want to access the Linux machine on campus, you can download and install mobaxterm. (http://mobaxterm.mobatek.net/).

Appendix B: Using sftp via command line (for your reference)

If you are not using mobaxterm, but only have a shell, you can run sftp via the command line:

In a mobaxterm shell or any other shell, type:

```
pwd
```

1. Navigate using the change directory (cd) command to a folder of your choice on the PC. Then, type:

```
sftp username@egr3.up.edu
```

Type in your password. Click the do not remember password.

You should see a prompt that looks like:
`sftp>`

2. Find your `hello.c` file by using the `cd` command. When the file is in the current directory, type:
`get hello.c`

3. Then, type:
`bye`

4. The file should be transferred to the folder you navigated to using the shell on the PC.