Lab 0: An Introduction to Linux and C

CSE/IT 113

NMT Computer Science

Lab 0 is provided as a reference material only. There is nothing to turn in for this lab.

1 Introduction

The purpose of this lab is to introduce you to the Linux environment (including some common, useful commands) and to the C programming environment. In this lab, you will learn how to

1. Navigate a file system
2. Manage directories from the command line.
3. Edit, compile, and run a short C program.
4. Create a Tar archive and submit a lab.

Throughout this semester's labs, we will be using specially formatted text to aid in your understanding. For example,

```
text in a console font within a gray box
```

is text that either appears in the terminal or is to be typed into the terminal verbatim. Even the case is important, so be sure to keep it the same!

```
Text that appears in console font within a framed box is sample C code
or program output.
```

Text in a `simple console font`, without a frame or background, indicates the name of a file or some action you need to perform, but not necessarily type into the terminal. Don't worry: as you become familiar with both the terminal and C, it will become obvious which is being described!

2 Navigating a Linux File System

2.1 shell

At the terminal prompt you are using what is called a shell. A shell provides a means to interact with the operating system.

2.2 File Hierarchy

Unix has a logical file system, which means as an end user you don't care about the actual physical layout and can focus on navigating the file system.

Key: Everything begins at the root directory `/` and all other directories are children of the root directory. The file system forms a tree.

A path consists of the names of directories and is a way to navigate from the root file system to the desired directory. If you include the root directory in the path this is called an absolute path. Another way to navigate the tree is to use relative paths, which navigate the tree based on where you currently are.

To navigate the file system, a couple of useful commands.

2.2.1 pwd - print working directory

print **w**orking **d**irectory displays the directory you are in.

```
$ pwd
/home/cse
```

The directory you are currently in is called your “working” directory.

2.2.2 cd - change directory

`cd` allows you to change your working directory to another location.

syntax:

```
$ cd <path_name>
```

where the <path_name> is either relative or absolute.

```
$ cd /usr/local/bin
$ pwd
/usr/local/bin
$ cd -
```

Where are you?

```
cd -
```

Now where are you?

`cd -` returns you to your previous working directory

The `-` is called a command line argument or option. To view the list of options for a command use the manual.

```
$ man cd
```

`man` is short for manual

Notice the man page comes up with a Bash built-in. Bash is the name of the shell and built-in commands are already loaded into memory.

To move up one directory (one node) level use dot-dot.

```
$ cd ..
```

Where are you?

To move up two levels use dot-dot slash dot-dot

```
$ cd ../..
```

To go to your home directory you can type

```
$ cd /home/<user>
```

where <user> is your user name or use a tilde

```
$ cd ~
```

or you can use

```
$ cd $HOME
```

where $HOME is an environment variable that stores the path to your home directory. This is more useful in shell scripting than path navigation. Use tilde instead.

2.2.3 ls - listing the contents of a directory

`ls` allows you to list the contents of a directory. In other words what files and directories are located there.

```
$ cd ~
$ ls
<a list of file names and directories contained with your working directory>
```

Adding some command options let you see other file information

```
$ ls -alFh
...
-rwxr-x---.  1 scott scott  1.0K Jan 12 14:56 down_and_out*
drwxr-xr-x.  5 scott scott  4.0K Jan 21 17:04 Downloads/
-rw-r-x---.  1 scott scott  2.2K Jan 12 11:31 .emacs
...
```

The `a` option show hidden files (aka dot-files), the `l` option produces a long listing format of file names and directories, the `F` option adds a `*` to the end of the file if is an executable and a `/` if it is a directory, and the `h` option prints out the size of the file in human readable format.

2.3 Other directory commands

```
mkdir
cp
mv
rm
touch
cat
less
script
```

2.4 Creating directories

Directories are very important for organizing your work, and you should get in the habit of creating a new directory for every lab this semester.

1. In the terminal, change your current working directory to the top level of your home directory by typing the following command:

   ```
   cd
   ```

2. Create a `cse113` directory by typing the following command;

   ```
   mkdir cse113
   ```

3. Change your current working directory to the `cse113` directory by typing the following command:

   ```
   cd cse113
   ```

4. Create a sub-directory for this lab in your `cse113` directory by typing the following command:

   ```
   mkdir lab0
   ```

5. Change your current working directory to the `lab0` directory by typing the following command:

   ```
   cd lab0
   ```

2.5 Using the file system

The following instructions will show you how to maneuver in the Unix file system. For more information on these commands, review the TCC publications "Summary of common Unix commands" (`http://infohost.nmt.edu/tcc/help/pubs/unixcrib/`) and "Customizing your Unix account" (`http://infohost.nmt.edu/tcc/help/pubs/dotfiles/`).

1. In the terminal, type the following command:

   ```
   pwd
   ```

   `pwd` stands for 'print working directory,' and the result is called the *absolute path* to this location. In this instance, the result should be something similar to

   ```
   /u/username/cse113/lab0
   ```

   where `username` is your TCC account username.

2. Type the following command:

   ```
   ls
   ```

   `ls` stands for 'list segments' (from Multics) and it prints to the screen, *standard out*, a list of files in the current directory. At this point, you should see nothing, because you have not put any files into your `lab0` directory. If, instead, you type the following command:

   ```
   ls -a
   ```

   you should see two entries: `.` and `..`, which exist in every directory. The `.` stands for 'this directory' and the `..` stands for the 'parent directory.'

3. Try the following commands to move within your directory structures:

   ```
   cd .
   pwd
   cd ..
   pwd
   ```

4. To return to your `lab0` directory, type the following command:

   ```
   cd lab0
   ```

5. Now type the following commands:

   ```
   cd
   pwd
   ```

   Typing `cd` without any parameters should take you to your 'home directory.'

6. Using the commands you've just learned, return to your `lab0` directory.

7. In your `lab0` directory, type the following command, and then find out where it takes you:

   ```
   cd ../..
   ```

8. From your current directory, type the following command:

   ```
   cd cse113/lab0
   ```

   Notice that you can use multiple paths at once on the command line, separated by `/`. If you've used Windows for some time, you will notice that this differs from Windows path designations where the directory separator is `/`.

3 Instructions

The following instructions will guide you through this lab. Because this is an introductory lab, the instructions are rather detailed and specific. However, it is important that you understand the concepts behind the actions you take—you will be repeating them throughout the semester. If you have any questions, ask the instructor, teaching assistant, or lab assistant.

3.1 Your First C Program

Make sure you are in your lab0 directory. Open gedit from the terminal

```
gedit&
```

Type in the folllowing "Hello world!" program.

```
#include <stdio.h>

int main (void)
{
  printf ("Hello, world!\n");
  return 0;
}
```

Save the file (in your lab0 directory) as hello.c

Return to your terminal and compile and run your code.

To compile,

```
gcc -Wall hello.c
```

Your code should compile without warnings or errors and you should now be able to run it.

To run a file,

```
./a.out
```

`a.out` is the name of the executable file that was created when you compiled your code. The ./ in front of it tells the compiler that you wish to execute this file and the file resides in the current directory (i.e. your current working directory).

You can easily change the name of the executable file that you create. Try the following steps:

Compile:

```
gcc -Wall hello.c -o hello
```

Run:

Since you have now saved your executable file to 'hello', you can run it using the following command.

```
./hello
```

3.2 Make a script

It is often useful to keep a log of what commands you are executing and their output. To do that, we use the `script` program.

1. To keep a log of your next actions, type the following command:

   ```
   script lab0.script
   ```

2. Type the following commands:

   ```
   gcc -Wall hello.c -o hello
   ./hello
   ```

   You are compiling and running the program again, but this time the compilation, your input, and the output from the program will be saved to the script file. Once your program returns you to the command line prompt, you can exit your script.

3. To end the script, type the following command:

   ```
   exit
   ```

4. Although the script is done, you need to add some extra information. Use a text editor to edit the script file you just created: add your first and last names, the lab number, and the date on different lines at the top of the file. It should look like the following when you're done:

   ```
   Firstname Lastname
   Lab 0
   Month Day, Year
   ```

5. Save the file and close the text editor.

3.3 ASCII Art

This is the fun part of this week's lab. ASCII Art!

Open up gedit or some other editor of your choice and save a blank document as ASCII.c

Add the basics for a C file:

```
#include <stdio.h>

int main(void)
{

  return 0;
}
```

Now you can use printf statements to make "art" using letters, numbers, and characters.

Make whatever you would like. I did a "K". Remember to put newline characters at the end of each line! Also, feel free to make more than one design. You have no limitations for this assignment.

After you have completed your designs, make a script of your ASCII art.

```
script ASCII.script
gcc ASCII.c -o ASCII
.\ASCII
exit
```

```
#include<stdio.h>

int main(void)
{
  printf("k        k\n");
  printf("k       k\n");
  printf("k      k\n");
  printf("k     k\n");
  printf("k    k\n");
  printf("k   k\n");
  printf("k  k\n");
  printf("k k\n");
  printf("kk\n");
  printf("k k\n");
  printf("k  k\n");
  printf("k   k\n");
  printf("k    k\n");
  printf("k     k\n");
  printf("k      k\n");
  printf("k       k\n");
  printf("k        k\n");

  return 0;
}
```

3.4 Creating Tar archives

Tar is used much the same way that Zip is used in Windows: it combines many files and/or directories into a single file. Gzip is used in Linux to compress a single file, so the combination of Tar and Gzip do what Zip does. However, Tar deals with Gzip for you, so you will only need to learn and understand one command for zipping and extracting.

In the terminal (ensure you are in your `lab0` directory), type the following command (ALL ON ONE LINE), replacing `firstname` and `lastname` with your first and last names:

```
tar -czvf cse113_firstname_lastname_lab0.tar.gz hello.c lab0.script
ASCII.c ASCII.script
```

This creates the file `cse113_firstname_lastname_lab0.tar.gz` in the parent directory. The resulting archive, which includes every file in your `lab0` directory, is called a tarball.