Prelab 0: An Introduction to Linux

CSE/IT 113L

NMT Department of Computer Science and Engineering

"Good code is its own best documentation."

— Steve McConnell

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

— Tony Hoare

"One of my most productive days was throwing away 1,000 lines of code."

— Ken Thompson

Contents

1 Glossary

absolute path: an absolute path makes no assumptions about your current location with the file system. It always begins from the root (/) directory and describes every step you must take through the file system to end up at the target location.

archive file: a single computer file that contains one or more files that have been compressed (or reduced in size).

compile: the act of converting programs written in a high level programming language (such as C), which is understandable and written by humans, into a low level binary language understood only by the computer.

console: a computer terminal where a user may input commands and view output on the monitor.

directory: a file system cataloging structure which contains references to other computer files, and possibly other directories.

executable: a computer file that contains an encoded sequence of instructions that the system can execute directly when a programmer invokes the run command.

file: the smallest unit (or container) in a computer system for storing information.

file system: determines the way files are named and how files are placed or retrieved logically in a computer.

Multics: the first operating system to provide a hierarchical file system.

pair programming: a technique where two, or three, programmers work together on a single computer. During a pair programming event, group members alternate between roles every ten minutes.

path name: the name of a file or directory together with its position in relation to other directories traced back in a line to the root or your current working directory.

relative path: a relative path describes the location of a file relative to the current working directory.

script: the `script` command makes a typescript (file with text in it) displayed on your terminal.

shell: the shell is the layer of programming that understands and executes the commands a user enters. In some systems, the shell is called a command interpreter.

standard out: the destination of all but the error-related output of a command; by default, this destination is generally the terminal.

tarball: a set of files packaged together into a single file, then compressed using the gzip compression program. Tar is short for tape archive, which tells you something about how long this particular mode of packaging files has been around.

terminal: a text-based interface to the computer.

2 Introduction

The purpose of this prelab is to introduce you to pair programming, the Linux environment, and some common & useful commands. In this prelab, you will learn how to

1. Navigate a file system
2. Manage directories from the command line.
3. Compile and run a short C program.
4. Create a Tar archive.
5. Submit a prelab.

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

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

is text that either appears in the **terminal** as output or is to be typed into the terminal verbatim, except the leading $ which is used to indicate the **shell** prompt. Even the case is important, so be sure to keep it the same!

```
1 Text that appears in console font within a framed box is sample C code
2 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.

3 Pair Programming

Over the span of this course you will be introduced to and take part in **pair programming**. Pair programming is a technique where two, or three, programmers(you and your partner(s)) work together on one computer. You each have unique roles:

- One person will be the *driver* while the other will be the *navigator*.

- The *driver* will be the one working on the computer, writing code.
- The *navigator* keeps an eye on the code, checking for typos or mistakes.

Sometimes where three people may be grouped together and in this circumstance the third person is the *keeper*.

- The *keeper* considers the overall design of the assignment and keeps track of upcoming problems and possibly how to improve on the code.

If this third role is not filled the *navigator* takes over these responsibilities. Switching roles is a key part of this technique. We will be using pair programming in all of the prelabs for this course, so you will have plenty of time to familiarize yourself with this technique.

When it comes time to submit your pair programming assignment only one partner needs to submit the **tarball**. This submission counts for all students in the group and the Canvas submission page will reflect this.

4 Navigating a Linux File System

4.1 shell

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

4.2 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 file system tree based on where you currently are.

To navigate the file system, some useful commands are listed below.

4.2.1 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.

4.2.2 cd - change directory

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

syntax:

```
$ cd <pathname>
```

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 ..
```

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. Or even easier, just type `cd`

```
$ cd
```

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

4.3 Other directory commands

Here is the list of some other directory commands:

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

Use the `man` command with these commands to explore what they do.

4.4 Creating directories

Directories are very important for organizing your work, and you should get into 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 prelab0
```

5. Change your current working directory to the `prelab0` directory by typing the following command:

```
$ cd prelab0
```

4.5 Using the file system

The following instructions will show you how to maneuver in the Unix file system. For more information on these commands, see *The Linux Command Line*.

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

```
/home/username/cse113/prelab0
```

where `username` is your username. However, depending on the file system set up on your machine, you may see something slightly different before the `/cse113/prelab0`. That's OK! As long as you see the `/cse113/prelab0` you are in the correct directory.

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 `prelab0` 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 `prelab0` directory, type the following command:

```
$ cd prelab0
```

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 `prelab0` directory.
7. In your `prelab0` 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/prelab0
```

Notice that you can specify a path through multiple directories 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 \.

4.6 Making 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, from your prelab0 directory type the following command:

```
$ script example.script
```

2. Type the following commands:

```
$  pwd
$  ls -a
```

3. To exit the script type:

```
$ exit
```

   Now you have a **script** named `example.script` which shows your current working directory and its contents.

4. If you ever want to view the contents of a file use the `cat` command. To view your script file type:

```
$ cat example.script
```

5 Requirements

For this prelab you and your pair programming partner must carry out the following exercises.

- Create a script showing you can navigate the Linux Command Line.
- Create a script showing compilation and play-throughs of the game you have been given.

For each exercise you will switch roles, *driver* and *navigator*, every 10 minutes in order to get familiar with pair programming concepts. It will become more useful as the prelabs become more complicated, for now we just want you to get some practice. Your instructor will have a timer that will signal when you should switch roles.

5.1 Navigating the Linux Command Line

Exercise 1 (commands.script).
Navigating the Linux Command Line Using the commands you have learned, perform the following tasks. Record your actions in a script called `commands.script`. Remember that the manpages contain useful information!

1. Begin a script called `commands.script`.

2. Create a folder called `commands` in your `prelab0` directory.
3. Navigate into `commands`.
4. Have the terminal print out your current working directory.
5. Create a text file (you can name it whatever you want).
6. Add some text to that file (again, you can type whatever you want). ***Note Bene:*** If you have VS Code installed you can use the `code` command followed by the name of your file to open that file in the VS Code text editor in order to type into the file.

   ```
   $ code <filename>
   ```

7. Print out the contents of that file to the terminal.
8. Copy the text file you created into the folder where you created `commands` (i.e. back one directory).
9. Move back one directory.
10. Print to the terminal the contents of that directory.
11. Delete the folder `commands` and its contents.
12. End the script session by typing `exit`.
13. Verify that the file `commands.script` exists in your current working directory and contains your group's terminal interactions by displaying it's contents to the screen with the following command:

    ```
    $ cat commands.script
    ```

5.2 Hangman

Exercise 2 (hm-out.script).
You most likely know about the Hangman game. To play the game, one player chooses a word and the other is given a finite amount of tries to guess the letters in the word. If they fail so many times they lose and the man gets hung. You have been provided with the C source file for this game. You must turn in a script file (covered below) that shows this program being compiled and run with at least 3 plays (win or lose, does not matter).

To compile and run the game make sure you are in your `prelab0` directory.

Now run the following command,

```
$ gcc -g -Wall hangman.c
```

The game should **compile** without warnings or errors and you should now be able to run it.

To run an executable file,

```
$ ./a.out
```

`a.out` is the name of the executable file that was created when you compiled the game. 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 -g -Wall hangman.c -o hangman
```

Since you have now saved your executable to a file named `'hangman'`, you can run it using the following command.

```
$ ./hangman
```

Now that you are familiar with compiling and running a program you can create a script called `hm-out.script`. Record the steps for compiling the program and at least 3 play-throughs in this script.

Compile and run 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.

- 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. The first three lines of the script file should look like the following when you're done:

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

- Save the file and close the text editor.

6 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 perform the same tasks that 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 `prelab0` directory) type the following command, replacing `NUMBER` with your group number.

```
$ tar -czvf cse113_groupNUMBER_prelab0.tar.gz commands.script hangman.script
```

This creates the file `cse113_groupNUMBER_prelab0.tar.gz` in the parent directory. The resulting **archive** file, which includes the `commands.script` and `hangman.script` files in your `prelab0` directory, is called a tarball.

Submitting

You should submit your code as a tarball file that contains all the exercise files for this lab. The submitted file should be named (**note the lowercase**)

`cse113_groupNUMBER_prelab0.tar.gz`

Replace the `NUMBER` with your assigned group number and have one group member upload the tarball to Canvas before the due date.

Only one person in your group needs to upload your group's tarball to Canvas. If, upon being uploaded, all members of your group do not see that the file has been submitted, please contact the instructor or head TA as soon as possible.

Upload your `.tar.gz` file to Canvas.

List of Files to Submit

Exercises start on page 9.