CS 103 Lab - Linux and Virtual Machines

1 Introduction

In this lab you will login to your Linux VM and write your first C/C++ program, compile it, and then execute it.

2 What you will learn

In this lab you will learn the basic commands and navigation of Linux, its file system, how to write, compile, and run C++ programs.

3 Background Information and Notes

3.1 Install the Course VM

Follow the course virtual machine installation instructions found at the link below.

http://bits.usc.edu/cs103/install

Budget a couple of hours for this as early as you can. In case of problems, visit the office hours of course staff.

3.2 Unix, Linux, Ubuntu, C, C++[1]

Unix was a commercial operating system developed by AT&T Bell Labs in 1969. At UC Berkeley, researchers developed it further, and Unix took off as the operating system of choice for developers who wanted to modify and extend the kernel (core) to fit their needs. Many variants and clones of the system were developed.

In 1991, a student developer named Linus Torvalds wrote a new variant and released it as *free software* (not free as in beer, but rather free as in freedom). Together with a suite of tools (compilers, editors, shells, etc) known as the GNU Project, it made it feasible for the first time to for a community of users to jointly develop software free of commercial interests and licensing. These days, almost all web servers run some variant of Linux. It is also starting to enter the mainstream, e.g. the Android operating system. Ubuntu is a "Linux distribution." It tries to collect all of the bits and pieces needed to make Linux easy to use: an installer, a user interface, a package manager, etc. OS X and Android are based on Unix/Linux.

C was developed between 1969 and 1973 jointly with Unix. Unix and Linux are primarily written in the C programming language. C++ is an extension of the C language developed in the mid-1980s to add object-oriented programming and many other features.

[1] **Acknowledgement**: Much of the material covered in this handout is taken from a tutorial produced by Bilal Zafar and user guides prepared by the Information Technology Services at USC.

4 Procedure and Reference

4.1 Starting your VM

Follow the link on the previous page and follow the instructions therein.

Note: In Virtual Machine terminology we refer to the 'host' OS which is the actual OS your PC is running (OS X or Windows) and the 'guest' OS which is the OS running virtually (i.e. Ubuntu).

4.2 File System and Navigation Commands

It is important to understand how directories are arranged in Unix/Linux. Logically, Unix files are organized in a tree-like structure (just like in Windows). '/' is the root directory (much like C: is for Windows). Underneath the root are other directories/folders with a sample structure shown below:

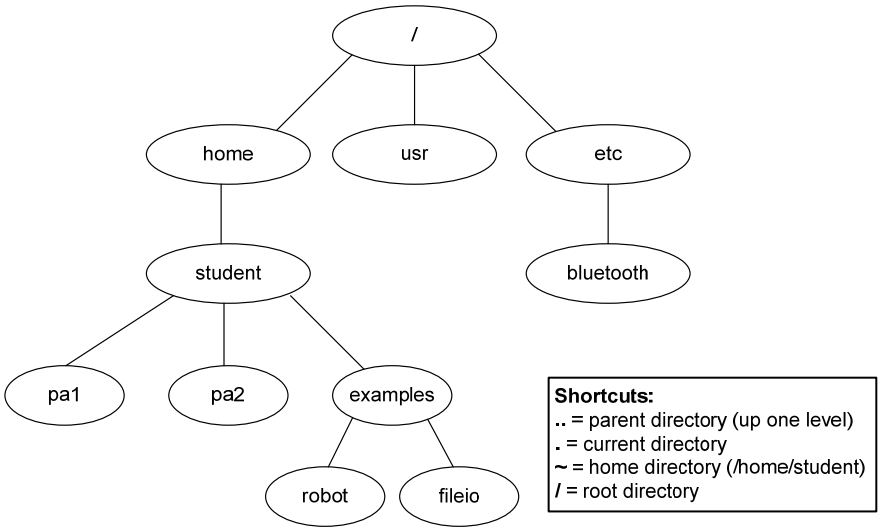


Figure 1- Example Unix/Linux File System Structure

The `/home` directory contains the user files for all accounts on the system. Other top-level directories include applications, libraries, drivers, et cetera. The name of the account you will use is `student` and so your files will be located in the `student` subdirectory of `/home`, which can also be referred to as `/home/student` or using the shortcut name ~ (tilde sign).

After you log into the system, click the icon shown at the left. This opens the Terminal, a text-based window for interacting with the system. It is also sometimes called the “command line.” Unlike Windows or OS X, you will primarily interact with the system by typing in commands. In fact, those systems have terminals too (called Terminal in OS X and Command Prompt in Windows). The commands you will learn are very similar to what you can do in OS X, and fairly similar to what you can do in Windows.

Here is a crash course to introduce three of the most important commands for text-based file system navigation:

- `cd`: change directory. At every moment, your Terminal window is "visiting" a particular folder called its current "working directory". The `cd` command is used to navigate from on working directory to another.
- `pwd`: report your present working directory (where you are now)
- `ls`: list the files directly inside of your present working directory

Enter these commands **exactly in the order shown below**. They'll show the variety of ways in which these commands can be used. You have to press Enter/Return after each command.

Type this:	Effect:
`cd /`	Go to the root directory of the file system
`pwd`	Prints **`/`** (present working directory is filesystem root)
`ls`	Prints the files and folders directly inside of `/`: `bin boot cdrom dev …`
`cd home`	Enter the home subdirectory of `/`. Note, the terminal shows **`student@course-vm:/home $`** which is just a reminder of who and where you are.
`ls`	Prints the one folder directly inside of `/home`: `student`
`cd student`	Enter the student subdirectory of `/home`.
`pwd`	Prints **`/home/student`** (you navigated to your home dir)
`ls`	Prints contents of your home dir (including `Desktop`)
`ls -al`	Prints hidden files and folders, and all file information. Note that it includes **`..`** which is a parent shortcut:
`cd ..`	Move to parent of current directory (back to `/home`)
`pwd`	Prints **`/home`** (you just navigated here)

`ls /usr`	Directly show contents of `/usr`
`ls /usr/gam` and then press Tab instead of Enter	The Terminal autocompletes `gam` to `games/` Autocompletion is extremely useful! Press Enter to list contents of `/usr/games`
`/usr/games/sol`	Run the `sol` program in `/usr/games` Close it whenever you're ready to move on...
`cd ~`	Return to your home directory
pwd	Prints **`/home/student`**

In general, you can use

`cd <directoryname>`

to enter a subdirectory of the current one,

`cd /<directoryname>/<directoryname>/`

to enter an absolute location, and you can always use ~ as a shortcut to your home directory and `..` as a shortcut to the parent or the current directory.

Many commands such as `ls` can either be typed plain or with "command-line arguments" like `ls -al`. Later in the course we'll learn about how to utilize these arguments in our own programs.

4.3 Editing and Organizing Files

Unix/Linux systems permit many text editors, though for beginners we recommend `gedit` which is simple and powerful (multiple tabs, search-and-replace, etc).

It's up to you how you want to organize your files on your virtual machine, but we recommend using a folder called `cs103` inside of your home directory (or inside of `Dropbox`). (Later in the semester you may want to organize your files into subfolders for each assignment.) Here's how to create a subdirectory and text file:

`cd ~`	Go to your home directory. (If you're using Dropbox for backup, use `cd ~/Dropbox` instead.)
`mkdir cs103`	Create `cs103` subdirectory
`cd cs103`	Enter it
`gedit hello.txt`	Start editing a new text file. Type something in and save. (To save, use the menu at the top of the screen **OR** the buttons just above the editing area.) While gedit is still open, try typing `ls` at the Terminal. It's frozen… we'll explain shortly. After, quit gedit and try typing `ls` at the Terminal. You should see `hello.txt` listed.
`more hello.txt`	Show what's inside of `hello.txt` (what you typed in).
`gedit hello.txt &`	Open hello.txt for editing **AND** (using &) let Terminal continue taking input. Edit and save. Don't quit gedit.
`more hello.txt`	See the changes you made. Afterwards, quit gedit.
`rm hello.txt`	Delete (remove) the file `hello.txt`.
`ls`	Now `hello.txt` is gone. (You'll still see the backup file `hello.txt~` that gedit creates automatically.)

Linux/Unix has many other useful commands:

- `man` (manual) which is the "help" command. Try running `man ls`
- `clear` which clears your terminal screen
- `rmdir` which removes a directory
- `mv` which renames or moves a file (an example is given further below)
- `cp` which copies a file (using the same syntax as `mv`)
- `ps` which lists all the processes currently running. Try opening `/usr/games/sol &` and then look at the output of `ps`.
- `kill` which terminates a program. Try killing the program number that `ps` listed next to `sol` and see what happens.
- `grep` which searches for text. What is the output of this command?
  `grep NAME /etc/os-release`
- `wget` which gets a file from the world-wide web. Try
  `wget ftp://bits.usc.edu/hi.txt`
  This copies a text file `hi.txt` from online. What's inside of it? (use `more`)

5 Procedure

We will now use the commands and knowledge discussed above to write and compile your first C++ program.

5.1 Writing your First Program

Start your Ubuntu VM if it is not already running.

Exercise: Write, compile, debug, and run your first C program.

Navigate into the cs103 folder you created earlier using the 'cd' command:

```
cd ~/cs103
```

(Or `cd ~/Dropbox/cs103` if you're using Dropbox.) Let's write our first C++ program. We will need to create and edit a text file that contains our source code. Do this with:

```
gedit hello.cpp &
```

Remember, the & sign allows the Terminal to accept commands while gedit is still open, which is convenient. In gedit, type the following program in verbatim.

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
   if (argc < 2) {
      cout << argv[0] << " expects a string to be entered";
      cout << " on the command line" << endl;
      return 1;
   }
   cout << "Hello " << argv[1] << ".  Welcome to CS103" << endl
   return 0;
}
```

Save the file and go back to the command window leaving gedit open so we can fix any errors if they exist. We now need to compile this program before we can run it. At the command prompt type:

```
compile hello.cpp
```

Notice an error is output from the compiler that a ';' is expected at the end of some line. That is because valid C++ statements end with a semicolon ';'. Add the

semicolon, save the file and repeat the compilation command. It should report no errors. Now run

```
ls
```

You should see a file named "hello" in the directory, which is the executable file that g++ produced. Execute the program by typing.

```
./hello
```

(The `.` is required for a technical reason.) You should notice that it complains that a string is expected. This is as indicated in our program, which wants the user to type in a string (in this case our name) on the command line after the executable file. Re-try the program with the following command:

```
./hello Tom
```

You should now see the expected output: "Hello Tom. Welcome to CS103." Congratulations!

We can now close gedit since we are done editing the program.

We have successfully created, compiled, and ran our program. It is recommended to keep your files organized nicely. Rather than cluttering up our 'cs103' folder, let us create an 'examples' directory underneath 'cs103' to put example code like this. Create the 'examples' directory:

```
mkdir lab-intro
```

Now let's move our files from the `cs103` folder to the `examples` subfolder. For this task we will use the `mv` (move) command which expects some number of source files followed by the destination folder.

```
mv hello* lab-intro/
```

The '*' is called a 'wildcard' operator and will match any files that start with hello and end with anything else.

Let's confirm the moved files. We'll list the contents of the `lab-intro` directory, but using path completion. Type `ls la` without hitting Enter and instead press Tab. It should complete to `ls lab-intro`; accept this and check that the .cpp and executable files are there inside of the `lab-intro` subdirectory.

6 (Optional) About the Compiler

The `compile` command is not standard in Linux/Unix. Instead, it is a version of a compiler called `clang++`. You can directly run the compiler yourself manually; to do so, enter the `cs103/lab-intro` directory, then run

```
clang++ hello.cpp
```

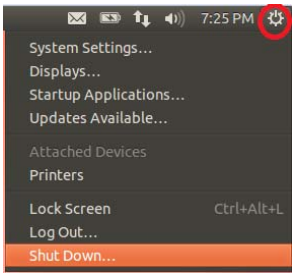
This will compile your file, however it will leave the compiled output in a file called `a.out` as its default action. This is ok but a little weird, so you may want to `rm a.out` and then, to make `hello` the executable name instead, run

```
clang++ hello.cpp -o hello
```

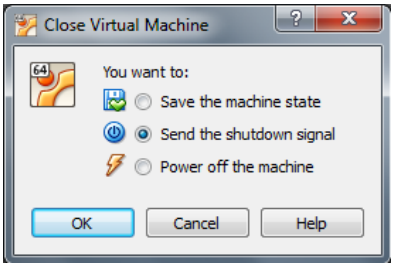
Now you can run `./hello` as before.

The `compile` command automatically calls `clang++` with many options (compiler flags), to enhance the quality of error messages, and to enable debugging. See http://bits.usc.edu/cs103/compile/ for more information. If you're not using our VM, replicating these flags is a very good idea; you might get different errors when submitting your code, otherwise.

7 Shutting Down



After completing the review exercises on the next page, you'll want to shut down your VM. You can do this from inside of the guest VM (see the picture at left). You'll be notified if you have any unsaved work.



You also quit using the host Oracle VirtualBox program (see the picture at right). "Send the shutdown signal" does basically the same thing as above. If you "Save the machine state" everything will be exactly the same when you start back up (sometimes called "hibernation") though it will use temporary disk space on your laptop. Avoid "Power off the machine", which yanks out the (virtual) power cord and can cause you to lose work.

8 Review

On the Coursework page,

http://cs103.usc.edu/coursework/

you will find a link to submit your answers to these review questions. Demonstrating that your VM is running in person is not required this week, but it is recommended so that you know where your lab is and can meet your TAs/CPs. Note that for all *future* weeks, it will be mandatory to check in your lab face-to-face.

Review questions:

1. What command will copy all files in the current directory to its parent directory?
2. If a program becomes unresponsive, what two commands could you use to identify the faulty program number and terminate it?
3. When you ran `grep`, the version of Ubuntu you're currently running was listed. What version is it?
4. When you used `wget` to get the `hi.txt` file, what magic number did it contain? (Seek help from course staff if your internet connection didn't work.)
   For the remaining questions, read the syllabus, available on the course website.
5. When is the written midterm exam and when is the programming exam?
6. What percent of an assignment is deducted, for each late day?
7. How many grace days do you have for Programming Assignments during the semester?
8. When homework is assigned, when are you recommended to do it?
   Please read the Academic Honesty section of the syllabus completely and carefully.
9. Who are you allowed to show your programming assignment and lab programs to?
10. Where can you ask questions related to coursework?