# CS 103 Lab - Command Line and Compilation

## 1 Introduction

In this lab you will login to Vocareum 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 command line fluency for Linux (Vocareum) and MacOS (Windows command are very similar with some slight differences), how to navigate the file system, how to write, compile, and run C++ programs.

## 3 Background Information and Notes

### 3.1 Ensure you can access Vocareum

You should receive an invitation to join Vocareum (www.vocareum.com) and access our CS 103 course where you can find Lab 1. It is critical you be able to login to Vocareum as all labs and programming assignments will be performed on this platform.

### 3.2 Setting up a Local Programming Environment

While all assignments can be completed in Vocareum, it can be very useful to have a programming environment installed on your local PC. This has speed benefits, often better editor and debug capabilities, and allows you to work if network connections are limited or not available.

http://bytes.usc.edu/cs103/tools

Again this is not necessary now, but can be useful to investigate when you have a few hours available in the first few weeks of class.

### 3.3 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, most servers run Linux and many popular OS have some derivation from Linux (MacOS,

[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.

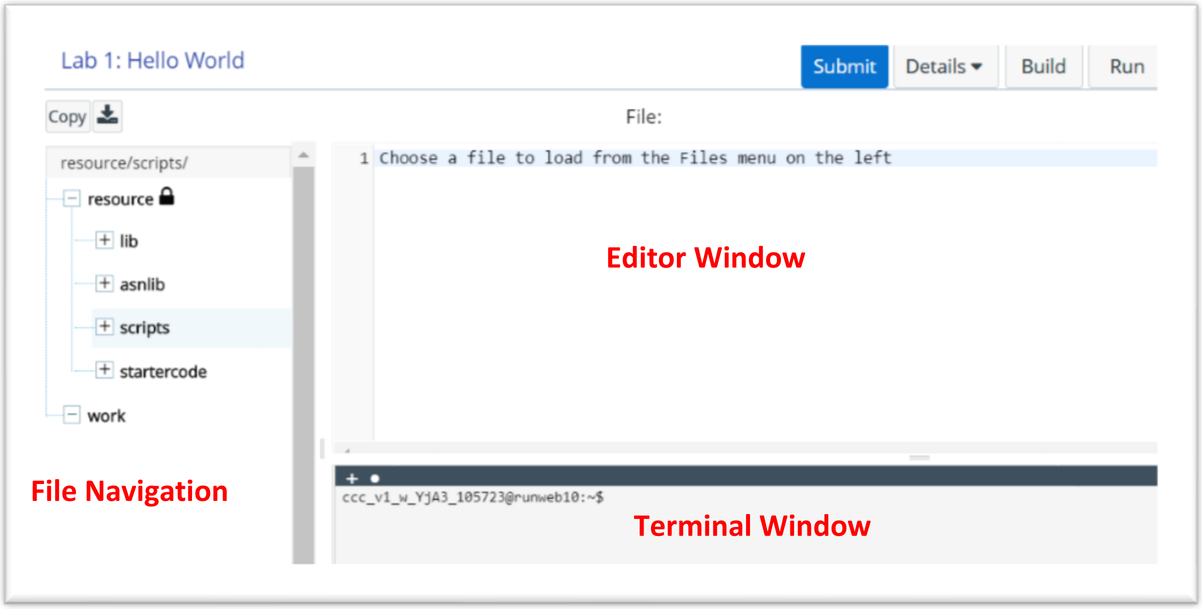
Android, etc.). The Vocareum website runs a Linux terminal where you can compile, run, and tests your programs.

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.

## 4 Procedure and Reference

### 4.1 Logging into Vocareum

Login to Vocareum (www.vocareum.com) if you are not already and find and click the Lab 1 link on the left-hand menu. After you click "Lab 1" you will enter the "workspace" for Lab 1:



```
Choose a file to load from the Files menu on the left
```

```
ccc_v1_w_YjA3_105723@runweb10:~$
```

Figure 1 - Vocareum Student View

You should see a 'work' folder in the left-hand file navigation area, an editor in the primary window, and a command line (aka "terminal" view on the bottom). **Click in the terminal area**.

### 4.2 File System and Navigation Commands

While this class may provide some shortcuts, future courses WILL require you to be adept at using the command line. And the terminal area of Vocareum is the same terminal/command line that all OSs provide. Thus, it is important to understand how to navigate the file system. Let's start by looking at how directories are arranged in Linux/MacOS (and fairly similarly in Windows). Logically, files are organized in a tree-like structure. '/' is the root directory (much like C: is for Windows). Underneath the root are other directories/folders with a sample structure shown below:

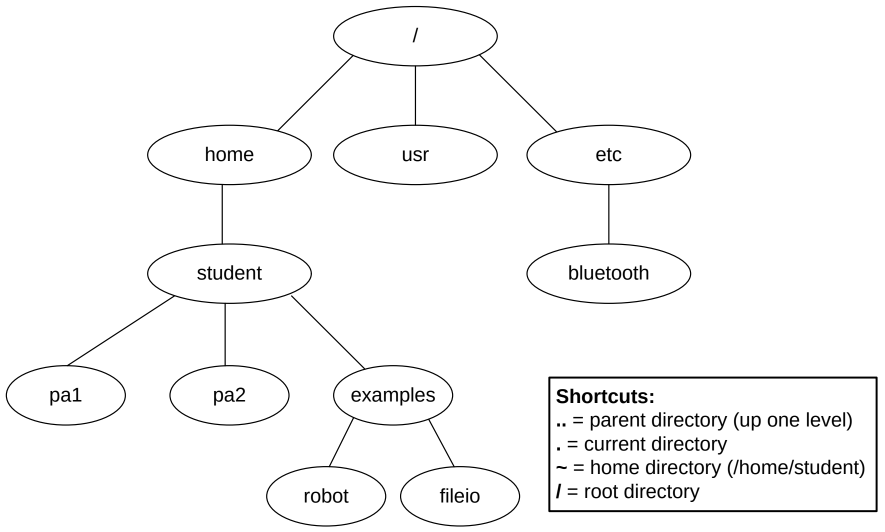


Figure 2- Example Unix/Linux File System Structure

The /home directory contains the user files for all accounts on the system. Other top-level directories include applications, libraries, drivers, et cetera. Vocareum will assign you some mangled user name like `ccc_v1_w_YjA3_105723` and assignment name (`asn80640_10/asn80641_1`) and then place all your work files in a folder that is aptly named `work`. So your files for this assignment will be located in a subdirectory several folders deep `/home/ccc_v1_w_YjA3_105723/asn80640_10/asn80641_1/work`. The shortcut name ~ (tilde sign) will always take you to your work folder

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: |
|---|---|
| `pwd` | Should print something like: `/home/ccc_v1_w_YjA3_105723/asn80640_10/asn80641_1/work` which is the directory you are in. |
| `cd /home` | Uses an absolute path (i.e. always starts from the root) to change to the root directory `/` and then into the subdirectory `home` |
| `ls` | Prints the files and folders directly inside of `/home`. You should see many folder names listed. |
| `cd ~` | Goes back to `/home/ccc_v1_w_YjA3_105723/asn80640_10/asn80641_1/work` since ~ is a shortcut to your work (home) directory |
| `ls ../resource/` | Uses a relative path (i.e. does NOT start with /) and the shortcut `..` to list the files one folder up from your work folder. You should see: `asnlib   lib   scripts   startercode` Note that these are the same folder you see under resources in the file navigate window on the left side of the Vocareum screen. |
| `cd ../resource/scripts` | Change directory to the scripts folder under resource |
| `ls -l` | Prints detailed information including modification date/timestamps for each file in the current directory |
| `cd ../../work` | Go back to your work (home) directory |

In general, paths that you type in at the command line for `ls` or `cd` (or other commands we will see in a second) can be:

- **absolute:** begin with **/** and are always relative to the root folder **/** (e.g. `/home/ccc_v1_w_YjA3_105723`**)**
- **relative**: do not begin with **/** and are relative to the present working directory (e.g. **`../resource`** which means go one folder up (**`..`**) from the present folder then down into a subfolder called **`resource`**).

### 4.3 Editing and Organizing Files

On your laptop you could use any editor (Sublime, VSCode, and Atom are just a few common ones that you can download). However, using Vocareum we have a built in editor.

To create, copy, and delete folders and files here are some commands:

| | |
|---|---|
| `touch` *`filename`* | Creates *filename* as a file in the current folder if it does not already exist<br>**Example**: `touch hello.cpp`<br>(creates hello.cpp) |
| `mkdir` *`foldername`* | Create *`foldername`* as a subdirectory in the current folder<br>**Example**: `mkdir cs103`<br>(creates a cs103 folder) |
| `cp` *`source_files dest`* | Make a copy of source file(s) to the destination folder.<br>*Note: * is a wildcard matching any filename*<br>**Example**: `cp *.cpp  cs103`<br>(copies all files ending in .cpp in the current folder to the cs103 subfolder) |
| `mv` *`source_files dest`* | Move or rename source file(s) to the destination.<br>**Example**: `mv hello.cpp bye.cpp`<br>(Renames hello.cpp to bye.cpp) |
| `rm` *`source_files`* | Deletes source file(s)<br>**Example**: `rm *.bak`<br>(deletes all files ending in .bak) |
| `rm -rf` *`foldername`* | Deletes the folder and all files/subfolders in it<br>**Example**: `rm -rf cs103`<br>(deletes the cs103 folder) |

Other useful Linux/Unix commands:

- `man` (manual) which is the "help" command. Try running `man ls`
- `clear` which clears your terminal screen

## 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

Ensure you are in the Lab 1 Assignment on Vocareum.

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

In the terminal, create a file:

```
touch hello.cpp
```

Refresh (collapse and expand the work folder in the left-side navigation area) and it should show the `hello.cpp` file. You could also use the Vocareum GUI to create a new file, but we want to focus on how the command line can be used. Click on the filename and the editor will load the file (though it will be blank). Type in the following program verbatim.

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

int main(int argc, char *argv[])
{
   if (argc < 2) {
      cout << "Provide your name on the command line" << endl;
      return 1;
   }
   cout << "Hello World!" << endl;
   cout << "Welcome to CS103, " << argv[1] << endl
   return 0;
}
```

The file will save automatically once you stop typing or click on the terminal. You should see check mark and the word "Saved" above the editor when the file has been saved. Go to the Terminal area and we will compile the program and to ensure no syntax errors exist. We now need to compile this program before we can run it. At the command prompt type:

```
g++ hello.cpp -o hello
```

This runs the compiler program (g++) and tells it the code to compile is in a file name `hello.cpp` and that the output produced by the compiler (i.e. the actual program) should be named `hello`. However, you will 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, by hitting the `Up-Arrow` button on your keyboard which will bring up the last command (you can cycle through all command by continually pressing the `Up-` and `Down-Arrow`). Use this shortcut to repeat terminal commands without having to type them again.

Are you rerun the g++ 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 name 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 by clicking the `Up-Arrow` to bring up the last command and then add your name (separated by a space) as in:

```
./hello Tommy
```

You should now see the expected output:

```
Hello World!
Welcome to CS103, Tommy
```

Congratulations!

We have successfully created, compiled, and ran our program. It is recommended to keep your files organized nicely.

One last shortcut exists, and it is for compilation. The g++ compiler has many options that can be added to the command line to have it allow it to check for additional sources of problems (warnings), enable debugging, and control optimizations for the code it produces. The command line we ran above was the simplest form of g++. We have created a script named `compile` which will add in many of those options. So you can also compile your code using the command:

```
compile hello.cpp
```

That command would have attempted to compile hello.cpp and created an executable program using the prefix of the filename (e.g. `hello`). If you had run

```
compile bye.cpp
```

It would attempt to compile a source file named `bye.cpp` and create an executable named `bye` (it uses the prefix of the filename and drops the .cpp)

## 6 (Optional) About the Compiler

As we said, there are many options to the g++ compiler. Here are a few basic ones that compile includes

```
g++ -Wall -g hello.cpp -o hello
```

The -g flag (option) includes debugging info in the executable so you can use a debugger like gdb (which you will learn in a later lab).
The -Wall flag (option) stands for Warnings: All and shows warnings which are issues in your code that aren't necessarily an error but may potentially cause problems and may need to be addressed. You should always read those warnings and ensure they are harmless.
Also, without the -o hello option, g++ will choose an output/executable program name on its own. It's default choice is the archaic name: a.out. So if you just typed

```
g++ hello.cpp
```

Your code would be compiled and an executable name a.out would be generated. It would work but it's not a very descriptive name. So it is good practice to use the -o name option.

## 7 Review/Checkoff

Please ensure you were successfully able to compile and run the "Hello world" program in a file name "hello.cpp". Failure to do so means you will not get credit for it.

Next, answer the review questions below by placing your answers in the `review.txt` file available in your work folder on Vocareum.

1. (true/false): I can submit programming assignments more than 2 days late.
2. (true/false): The penalty for each day late is 25% of the maximum points.
3. (true/false): I have 2 grace days for programming assignments where I will not have any penalty for submitting late.
4. (true/false): I am allowed to describe my code for a lab or assignment to my friends in CS 103.
5. (true/false): I can demo labs early but there is no late submission for labs.

Be sure to press the 'Submit' button in the upper-right of the Vocareum workspace to submit your code and answers and record your grade. Failure to do so will lead to a 0 for this lab.