CMPT 115: Principles of Computer Science

Lab1: Intro to the Command-Line, and Compilation

Department of Computer Science
University of Saskatchewan

January 11 – 15, 2016

Laboratory 1 Overview

Part 0	: Pre-Lab Reading
Part 1	: Logging In and Becoming Familiar with the Lab
Part 2	: An Introduction to the UNIX Command-Line
Part 3	: Compiling C++ Programs with the Command-Line
Hand in	: A transcript of your work with the command-line.

Part I

Pre-Lab 1 Reading

Mac labs

We're in the Mac lab because

- Keep things simple by supporting one platform
- Mac OSX is based on UNIX
- Windows is not based on UNIX
- UNIX is a system built by programmers for programmers.
- The most important work we'll do can be done on Mac or Linux with only a few minor changes.

Command-Line: Background

- Modern computer systems use “graphical user interfaces” (GUIs) with drop-down menus and mice, or touch-screens.
- Prior to the widespread use of GUIs, interaction with computers only used text!
- A text-based interface is called a *command-line*.
- Using a command-line is a repetition:
    1. User types the name of a command, followed by the RETURN key.
    2. Computer runs or “executes" the command.
    3. The computer shows that it is ready to accept another command.

UNIX Command-Line: What can it do?

- Everything you are used to doing with a GUI system can be done with a command-line, and more besides!
- For example, the command-line can be used as a file manager:
  - create files and folders
  - show file and document contents
  - copy and move files around
  - upload/download files from servers
  - search for information in folders, files, and documents
  - send documents to the printer
  - permanently delete files and documents
- There are no menus to list the commands, so the user needs a reference manual to look-up all the commands. Common ones were memorized just through use.

UNIX Command-Line: Why bother?

- The UNIX command-line is very useful and powerful for programmers, and advanced computer users.
- The UNIX command-line has a number of basic built-in commands, but the command-line interface gives you access to hundreds of utility programs written by programmers for programmers and made part of the standard UNIX tool-set. These tools are very often written in C/C++.
- For example, you can sort text files, extract data from documents, build C/C++ applications, typeset documents, plot graphs, fetch webpages, etc, etc, etc, all from the command-line.

UNIX Command-Line: The interface

- The command-line interface is simply a window for text.
- When the computer is ready to run a command, it will display a *command prompt*.
    - To issue a command, you type the name of the command, followed by the RETURN key.
    - The computer ‘runs’ or ‘executes’ the instructions, and when it is finished, it will display the command prompt again.
- The format of the command prompt varies from system to system.
    - It might show your your NSID, or some other information.
    - UNIX command lines often have the character $ or % near the end of the prompt.
    - Once you have learned a bit more, you can change the prompt to be whatever you like.

UNIX Command-Line: The context of a command

- This concept is very important. Don't dismiss it as trivial!
- A command is almost always a single word, or an acronym, related to the purpose of the command.
- The *context* for a command is the environment in which you type the command.
- The context is not represented visually, so you have to be aware of it, and keep it in mind.
- An important aspect of context for a command is the *folder* or *directory* in which you are working. This is known as *the current working directory*.
- Note: The words *folder* and *directory* refer to the same thing; *folder* is more modern, but the command-line often uses the older term *directory*. We'll write *folder* unless we are referring to a command that uses the term *directory*.

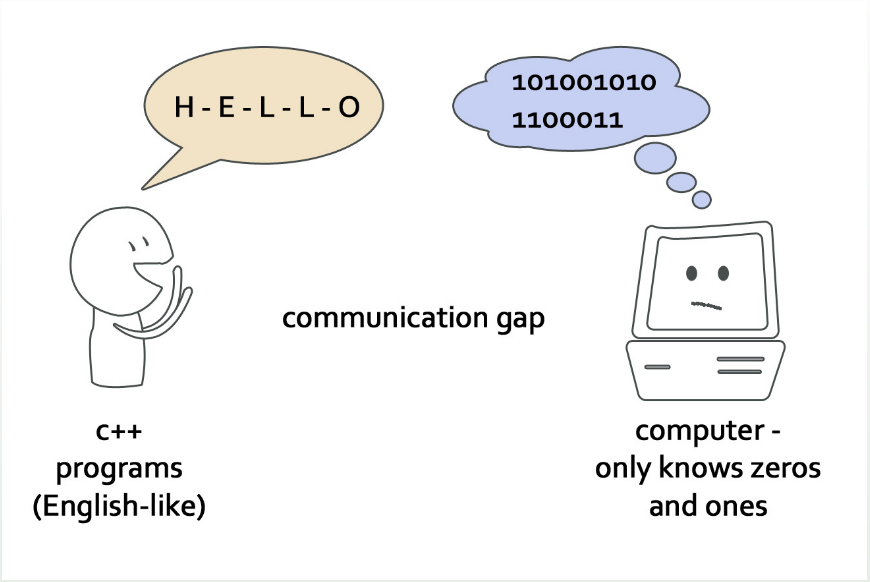
Paths, relative paths, and absolute paths

- In UNIX, a *path* is a sequence of folder names, separated by '/', that describes the location of a file or a folder.
- A *relative path* is a path that starts from the current working directory (so the path will describe a different location if you change working directories).
- An *absolute path* is a path that starts from a fixed, known location. An absolute path does not depend on the current working directory at all.

C++ in CMPT 115

- In 115/117 labs, we will continue to use C/C++.
- We will learn to use the command-line to “compile” and “run” your program (instead of Eclipse).
- C++ program files should have a “`.cpp`” extension (or “`.cc`”).
- We will demonstrate how to build (or “compile”) C++ programs using the g++ compiler.
- (Eclipse also uses the g++ compiler; we will learn how to use it directly, so you can use both!)

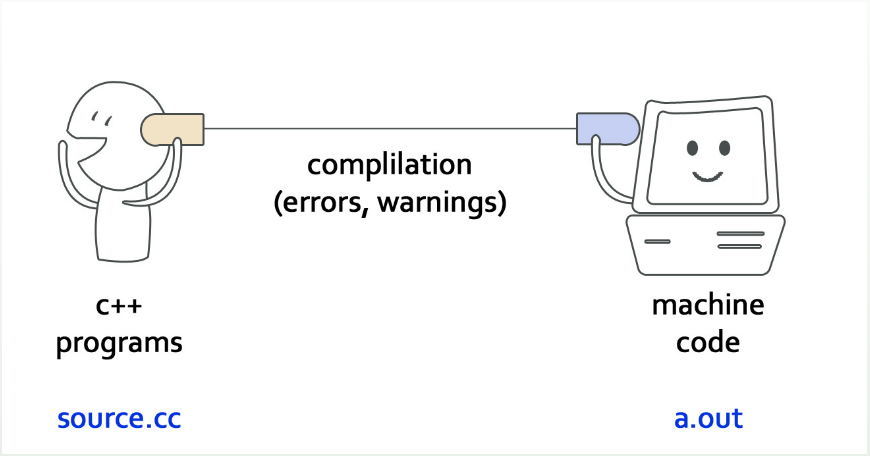
Compilation



Compilation - bridging the gap

- A *compiler* converts your "source code" (written in C++) into an application (machine language).
- The compiler is itself an application (and was also written in C).
- The input to the compiler is a C++ *source file*; in CMPT 111/116, you used Eclipse to create source files. In CMPT 115/117, you can use any editor (e.g., TextWrangler).
- The output of the compiler is a real application! It is stored in a file called `a.out` by default.
- If there are errors in your source file, the compiler will display error messages and warnings, and may not be able to build the application at all!
- If the compilation was successful, you can run your application by typing its name on the command-line.
- We will demonstrate all these ideas in the lab.

Compilation



Eclipse vs g++

- If you've used Eclipse, you may be used to some very nice features!
- We are not taking Eclipse away from you; we are adding the command-line to your skill-set. Feel free to use both. But you have to practice both to gain mastery!
- Eclipse is essentially a "manager". It manages editing, compiling and running.
  - The compiler is g++, the same as the command line interface. When you "build" a program using Eclipse, Eclipse simply commands g++ to compile it!
  - In Eclipse, when you click the "Run" icon, runs the app built by g++. Eclipse just displays a nice picture for you to click on to start it.
  - Eclipse also understands error messages produced by the compiler, and tries to highlight them in your program. This is something that TextWrangler does not do.

Part II

Logging In and Becoming Familiar with the Lab

Mac Software you'll want to know about

- Finder (file browser)
- Safari, FireFox (web browser)
- TextWrangler (programming editor)
- Preview (document & image viewer)
- Terminal (UNIX command-line)
- Activity Monitor (system browser)

Mac Software you'll not need

- Mail (use webmail instead)
  Too much trouble to set up for one course
- Xcode (use Terminal)
  Specific to Macs
- Microsoft Word
  Word documents are full of junk that we don't want in our programs

Part III

An Introduction to the UNIX Command-Line

UNIX Command-Line: Getting started

- On the Mac and on Linux, the command-line interface is essentially the same, as both systems are variants of UNIX.
- Windows also has a command-line, but the default interface is not UNIX, and is significantly different from Linux and Mac.[1]
- **Mac**: open Finder, go to "Applications", "Utilities", and run "Terminal".
- **Linux**: Right-click on the background, choose "Konsole."

ACTIVITY: Open your command-line! Note: Keep the Terminal window open until you are completely done with the lab. You'll copy/paste all the text in the window and upload it to Moodle. Further instructions at the end of the lab slides.

[1]You will not be able to use the Windows commands here, or the UNIX commands on Windows, unless you install something like Cygwin.

Determining the working directory with pwd

Activity

- To find out the directory (or folder) in which you are currently working, type pwd at the command prompt, followed by the return key.

```
% pwd
/home/abc123
%
```

- The command pwd[2] is an abbreviation for “print working directory.”
- The computer responds with a string that represents the path of the working folder or directory.
- When you type pwd on a computer in the Spinks labs, you should see a path that has your user ID on it. It may appear different on your own Mac or Linux computer.

[2]The command will not normally appear in red on your computer; we’re highlighting it to draw your attention to it in these notes.

Listing the contents of a directory with `ls`

- On the command-line, you can list the contents of a folder with the '`ls`' command.
- ACTIVITY: Use the `ls` command now, by typing it in the command line interface window at the command prompt.
- By default, '`ls`' lists the contents of the current working directory. Depending on your context, you will see different contents.
- Most commands have a default behaviour, but you can modify the behaviour by adding 'options' to the command.
- For example, '`ls`' does not show hidden files[3] by default.
- ACTIVITY: Type '`ls -la`' to reveal hidden files and extra information — notice the special folders and '.' ("dot") and '..' ("dot dot").

[3] In UNIX, a file is hidden only for tidiness. The hidden files are not secret files! Hidden files in UNIX are typically configuration files, and other meta-information.

Creating Folders with `mkdir`

- The command `mkdir` creates a new folder in the current working directory.
- ACTIVITY: Type `mkdir cmpt115` to create a new folder named "cmpt115".
- ACTIVITY: Use `ls` to check if the folder was created!
- Note: Spaces are meaningful to the command-line. If you type the command `mkdir cmpt 115`, you'll get two new folders, not one with a space!
- `mkdir` is an example of a command that requires an *argument*, i.e., some extra information.

Changing Folders

- An important aspect of a command's context is the folder in which the command is issued.
- It is possible to "move" to a different folder, and the new folder will be the context of commands that follow.
- On the command-line, this can be done with the command '`cd`'.
  - The command is an acronym for *change directory*.[4]
- ACTIVITY: Change the folder in the Terminal to the new folder you created earlier, by typing '`cd cmpt115`'.
  - This is another example of a command that can take an argument.
- Type '`pwd`' to verify that it changed successfully.[5]

[4] Acronyms are very common, because they are short and quick to type.

[5] On the Mac, if you type 'cd ' (notice the space!) without return, and then drag a folder from the Finder onto the Terminal, it will insert the path of that folder and you can type return to switch to it.

Creating new files

ACTIVITY:

1. Open the text editor (TextWrangler)
2. Type some text into the editor window. It doesn't matter at all what you type here!
3. Save the text as a file named 'lab1file.txt' in the new 'cmpt115' folder.
4. On the command-line, use the command '`ls`' to verify that it is there.

. . . and more!

We can scroll through a text-file with the 'more' command.

ACTIVITY:

1. Type '`more lab1file.txt`' will scroll through the file you created earlier.
2. You should see all the text you typed.
3. If you typed more than can be seen in a single window, 'more' will limit to the display to what fits in the window. To see more[6] of the file, press the space bar.

[6] Probably a pun.

More on context

- When we typed '`more lab1file.txt`' we only referred to the name of the text-file.
- That's because the command-line's context is the current working directory.
- If we want to access a file outside the current working directory, we need to know where the file is (more on this later).

Current folder and Parent folder

- Every folder contains two special folders, '.' (dot) and '..' (dot dot).
  - The first refers to the "current folder".
  - The second refers to the "parent folder".
  - The file '.' is simply the UNIX way for a folder to refer to itself (e.g. it's as useful as the English word "me").
  - Similarly, the file '..' is simply a way for a UNIX folder to refer to the folder it's in (e.g., it's like saying "the room I am currently in" but much shorter).
- ACTIVITY: On the command-line, type '`cd .`' then check the path. There should be no change!
- ACTIVITY: On the command-line, type '`ls .`'. You should see the files in the folder. The default behaviour of `ls` with no argument is the same as `ls .`
- ACTIVITY: Now type '`cd ..`' Check the path again. List the contents of the current directory.

Paths, relative paths, and absolute paths (recap)

- In UNIX, a *path* is the a sequence of folder names, separated by ‘/‘, that describes the location of a file or a folder.
- A *relative path* is a path that starts from the current working directory (so the path will describe a different location if you change working directories).
- An *absolute path* is a path that starts from a fixed, known location. An absolute path does not depend on the current working directory at all.

Paths, relative paths, and absolute paths

ACTIVITY:

1. Inside your 'cmpt115' folder, make a new folder called 'lab1' using Finder.
    - Open the Finder, go to the 'cmpt115' folder, make a new folder by right-clicking inside it, select 'new folder', and give it the name 'lab1'.
    - Right-click the 'lab1' folder, select 'get info' to check the path.
2. On the command-line currently examining the 'cmpt115' folder, we can switch to examining the new 'lab1' folder by using a path relative to the current one.
3. Type '`cd lab1`'. This is possible because the 'lab1' folder is directly inside the 'cmpt111' folder.

History of commands

- Typing lots of commands can become tedious.
- Modern UNIX command-line interfaces save previous commands that you typed, and allow you to reuse them quickly.
- ACTIVITY: Type the up-arrow and down-arrow to cycle through the history. From there it is possible to use directly, or edit a previous command. Experiment with the left and right-arrow keys as well!
- There is a lot to learn, but it's easy and once you master it, it is very powerful.

About spaces in document names, and folder names

- Files and folders are allowed to contain spaces.
- However: with the command-line, a space is used to separate different parts ("arguments") of commands. This creates an ambiguity in the possible meanings for spaces in a command.
- The UNIX system always treats a space as a separator, *unless the space is preceded by a backslash* '\'. The backslash[7] tells UNIX that the space is not to be used as a separator, so it can be part of a file or folder name.
- ACTIVITY: Make a new folder called 'test folder' and change to it. Don't forget the backslash!
- Generally, until you are reasonably familiar with the command-line interface, it's wise to avoid files and folders with spaces in the names. Use the underbar character '_' instead!

[7] Careful! The backslash and the forwardslash mean very different things!

Summary

We will introduce other commands as we need them.
In this part, we started to manage files using the command-line:

1. In the default directory (home directory), we first created a folder named ‘cmpt115’;
2. under this cmpt115 folder, we created a file (using a text editor) named ‘source.cc’;
3. then we created another folder named ‘lab1’ under ‘cmpt115’

Commands we used in this part

1. `pwd`
2. `mkdir`
3. `cd` (`cd.`) (`cd..`)
4. `ls` (`ls -l`) (`ls -la`)
5. `more`

Part IV

Compiling C++ Programs with the Command-Line

Our first program

ACTIVITY: Here is our first program. Create source.cc using a text editor (eg. TextWrangler), then type the code below into the file and save the contents in your 'lab1' folder. Make sure to change directories in the command line interface to that folder.

```
// Your Name Here
// Your Student Number Here
// Assignment or Lab Information Here
// Synopsis: Our first C++ program

using namespace std;

#include<cstdlib>

// main program entry point
int main(void){

      return EXIT_SUCCESS;

}
```

Notes on your source code

- In the previous slide, notice the comments at the top of the program.
  - Put your own name, Student number, etc in your file.
  - Remember where you save this file! You can use it as the base for many programs that you write this term.
- Also note the opportunity to describe to the reader what your program does.
  - Give the assignment or lab number.
  - Also, a one sentence or less description of what the program does.
  - This will cost you a few minutes, but will save you more time later. It's a good habit practiced by professionals.

Compiling a C++ Program

- ACTIVITY: Type the following into the command-line interface:

```
g++ -Wall source.cc
```

- This executes the g++ program (which is the C++ compiler) on the file 'source.cc'.
- The command put the *flag* `-Wall` to ask the compiler to give all possible warnings. It helps with debugging.
- The compiler creates an application based on your source code. The application is in a new file called `a.out`.
  ACTIVITY: Check that it is there with 'ls'.
- ACTIVITY: You can run your application by typing `./a.out` on the command-line. This basically says "run the `a.out` application found here in this folder". Your small app has become a command!
- Your application will do nothing, because the source code for the program was basically empty.

More practice with compilation

The program `printnumbers.cc` can be found on Moodle. Save that program in your 'lab1' folder, then switch the command-line to that folder.

```
// Prints out some numbers.

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

// main program entry point
int main() {

        int i = 42;
        float f1 = 3.14159, f2 = 1.0;

        cout << "\nHere's the answer: " << i
             << "\nThe cake is a lie: " << f1
             << "\nYou didn't lose, you " << fs << "!\n";

        return EXIT_SUCCESS;
}
```

Compiling a C++ Program

- ACTIVITY: Type the following on the command line:

```
g++ -Wall printnumbers.cc
```

- This executes the g++ program (which is the C++ compiler) on the file 'printnumbers.cc'.
- The `-Wall` flag is used to ask the compiler for as much checking and warning as possible. It helps with debugging.
- The compiler creates a working program based on your code. The program is in a new file called `a.out`. Check that it is there with 'ls'.
- You can then run the program by typing `./a.out` at the command prompt. This time, something a little more interesting happens.
- We will talk more about compilation in next lab.

Debugging a C++ Program

- The compiler will detect and report any errors found during the compilation process.
- If your source code has errors, the compilation process will not produce an application. Any errors found will be displayed in the command-line window; it may be very messy and confusing at first.
- ACTIVITY: In the source code for `printnumbers.cc`, remove the semi-colon at the end of the 'return' statement, save it and try to compile again.
- Notice that error messages appear in the command window. The compiler is trying to tell you that something is wrong. It uses line numbers and sometimes code snippets to show you where it got confused.
- From there, we can attempt to fix mistakes in the code, save, recompile, and if that is successful, run the program.

Debugging advice

- Start with a working program. Repeat the following:
    - Deliberately introduce an error into your program.
    - Use the command-line to compile the (now broken) program.
    - Observe the messages created by g++. Try to relate the message to the error you know caused it.
    - Restore the program by fixing that error.
- Keep a journal (a text file!) of all error messages you ever get from g++, and the causes that produced it.
- Try to introduce 2 or 3 errors at a time. Record the error messages and their causes.
- Building up your debugging skills is the best defense against frustration. Once you get a certain level of expertise, most bugs are simply annoyances, not catastrophes.

Summary

In this part, we learned how to compile and run a C++ program, using the following commands:

- If your program is called myprogram.cc, you can compile it with

```
g++ -Wall myprogram.cc
```

- You can then run your program with

```
./a.out
```

Part V

Lab 1: What to hand in

What to hand in

1. Open TextWrangler (or any editor you want)
2. Keep the Terminal (command-line) window open. Select all the text (Command-A), copy it (Command-C),
3. Move to the TextWrangler window; paste (Command-V) the work you did in the lab into TextWrangler. Save it as `Lab1_transcript.txt`.
4. Upload `Lab1_transcript.txt` to Moodle as part of Assignment 1.
5. Grading:
    - If you've used all the commands marked as ACTIVITY, in this lab, you'll get full marks.
    - If you hand nothing in, you'll get zero marks.
    - Your transcript will probably show evidence of commands being used incorrectly if you've misunderstood something. That's perfectly fine; no marks will be deducted.