Command Line Navigation and Compiling

You should become familiar with how to compile and work from the command line. Below is a quick reference and instructions on how to use the gcc c++ compiler (g++) from the Windows command line. g++ is included as part of the QTCreator bundled download. If you are on a Mac/Linux box, some commands will differ, but many will be the same.

Contents

Opening a command prompt:

Windows : You can use either the traditional Command Prompt or PowerShell. To open, go to the start menu and search for either "Command Prompt" or "PowerShell".
You can shift-right click inside a folder in Windows and get an option to **"Open command prompt here"** (may be "Open PowerShell window here" on more recent versions of Windows)

Mac : Use spotlight to search for **Terminal** and open a terminal window. From in finder you can also open a Terminal window at the location you are viewing.

Command line quick reference:

Goal	Command	Example:
Switch to different drive *Windows only*	*DriveLetter*:	E:
Change into a subdirectory	**cd** *directoryname*	cd cs161
Go up a directory level	**cd ..**	
Go to top level of this drive -For Mac/Linux	**cd ** **cd /**	
See what is in this directory -For Mac/Linux -Mac/Linux see more info	**dir** **ls** **ls -la**	dir
See only files of certain type -For Mac/Linux	**dir** *****.*extension* **ls** *****.*extension*	dir *.cpp ls *.cpp
See the contents of a file -For Mac/Linux	**type** *filename* **less** *filename*	type main.cpp less main.cpp
Compile a file *SEE G++ SECTION*	**g++** *filename* **-o** *program.exe*	g++ main.cpp -o myprogram.exe
Run a program -For Mac/Linux *PROGRAM MUST BE ON PATH OR IN CURRENT DIRECTORY*	*programName.exe* *./programName.exe*	myprogram.exe ./myprogram.exe
Windows Path:		
See PATH *SEE PATH SECTION*	**echo %PATH%** (Command Prompt)	

Speeding things up:
In most places times, pressing **tab** will try to autocomplete what you have typed so far. Hit tab multiple times to cycle through possible completions.
Use **up arrow** and **down arrow** to reuse recently typed commands.

Path (Windows):

Mac and Linux also use a PATH, but you should not need to edit them by hand.

The path is the collection of directories where Windows looks for programs when you try to execute a program with a command like "main.exe" or "g++". It is stored in a variable called PATH - it is just a list of directories separated with semicolons:

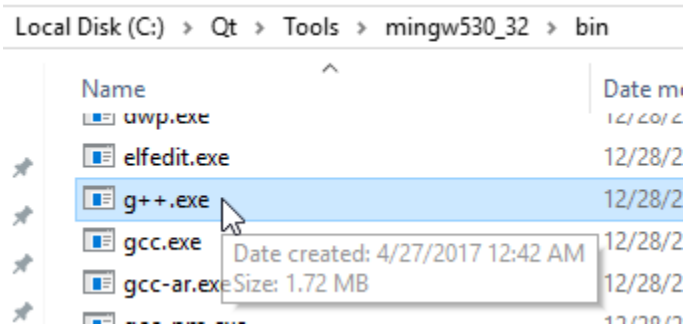
```
C:\Program Files (x86)\Java\jdk1.7.0_07\bin;C:\Cygwin\bin;…
```

To see your current path, type "**PATH**"

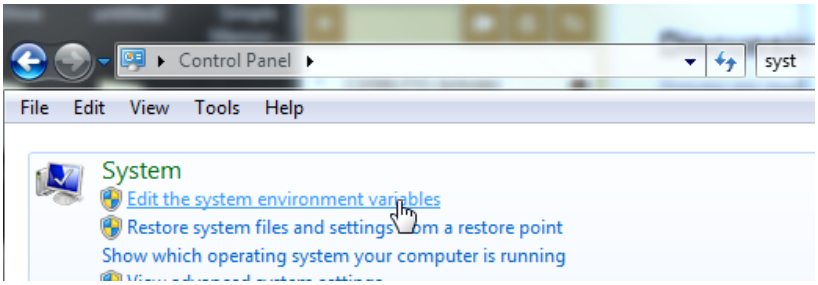
```
C:\Windows\System32>PATH
```

The path always includes the current folder. That is where Windows will look first for the program you tried to run. If it is not found, each directory in the PATH is checked to see if the program is there.

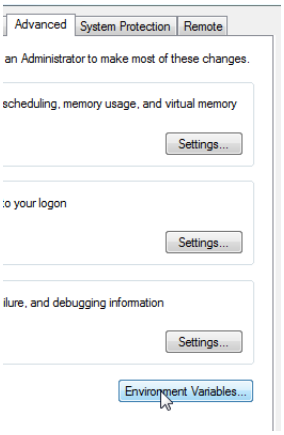
To get the c++ compiler working you need to make sure the folder containing g++ is on your path. It will be in the QT folder in a directory like `C:\Qt\ \Tools\mingw530_32\bin`. The exact version number (530_32) may be different, but it should be in a similar place (C:\QT\ Tools\mingwxx_xx\bin). Make sure you see **g++** in that folder. This will be the folder you need to add to the path:



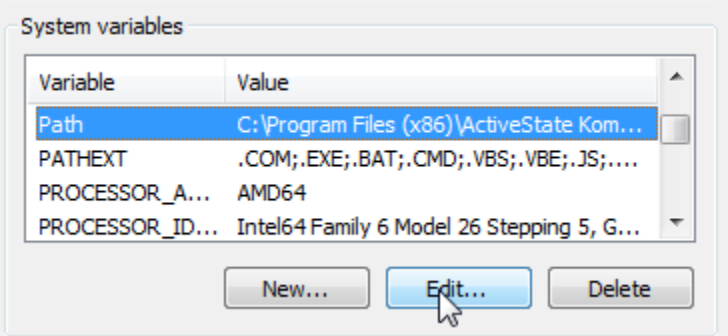
To permanently add this folder to your path so every command prompt window can make use of g++ automatically, you need to change the environmental variables. Open the control panel and search for "system" or "environment", find the **Edit system environment variables** panel:



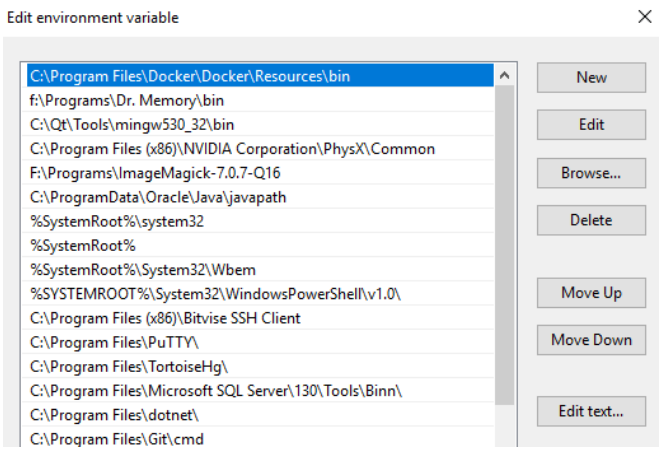
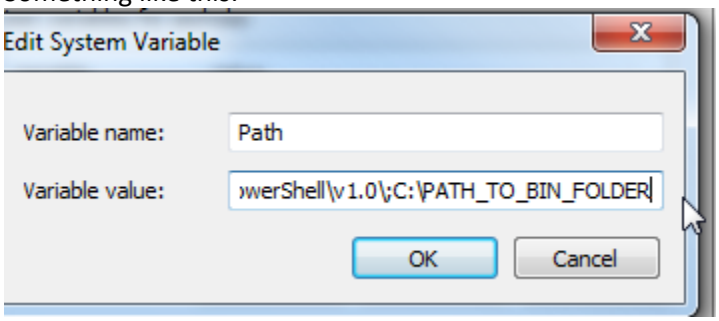
Then click Environmental Variables.



Under the System variables section, find **Path** and edit it:



The way to edit the path depends on how recent your version of Windows is:

Newer (shows path in window below)	**Older (shows path in window below)**
This window the path up into a list of folders. Simply click **New** and then type: **`C:\Qt\Tools\mingw530_32\bin`** (or wherever your g++ is located) 	Scroll to the end of the variable value and add **`;C:\Qt\Tools\mingw530_32\bin`** (or wherever your g++ is located) The semicolon is important to separate the new folder name from everything that comes before it. Something like this:  You may find it easier to copy the text out to notepad, edit it there and then paste it back in.

Using g++:

Windows: To run g++ you must tell Windows where to find it - see **Path** section above first.

Mac: The compiler you will use is called clang. But, it sets itself up to "hijack" calls to **g++**
So you can still just type **g++** or you can type **clang++**.

g++ run by itself will tell you "no input files"

```
C:\Users\Andrew\Desktop\CommandLinePractice>g++
g++: fatal error: no input files
compilation terminated.
```

g++ -c *filename.cpp* will compile your file into an object file *filename.o*
The –c says "Compile only", do not link into a final executable program. Try this and then look for the resulting new file.

```
ce>g++ -c simpleMain.cpp
```

g++ *filename.o* will link your object file with any libraries or other code it needs and produce an executable ***a.exe***. Try this and look for the new exe

```
ice>g++ simpleMain.o
```

Technically we are using the linker program "ld"… but g++ calls it for us with some extra options set

To run your program, type **a.exe** *On Linux/Mac, type **./a***

```
e>a.exe
```

g++ *filename.cpp* will assume you want to compile and link your object file in one step to directly produce an executable *a.exe*

```
ce>g++ simpleMain.cpp
```

To give your executable file a better name than a.exe, use the -o flag:
g++ *filename.o* -o *executablename*
Something like:
g++ main.o -o myProgram.exe

```
ce>g++ simpleMain.o -o myProgram.exe
```

To go directly from cpp file to final program, (compile and link code all at once), do:
g++ main.cpp -o myProgram.exe

```
e>g++ simpleMain.cpp -o myProgram.exe
```

This performs all the steps of compiling main.cpp. Then you can run your program by typing "myProgram.exe" *On Mac/Linux don't forget to put ./ before the program name.*

```
e>myProgram.exe
```

For more complete details on compiling from the command line, refer to section 1.3-1.5 of this reference:
http://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

g++ Options:

g++ supports many options. Simply include them after g++:
`g++ -g -Wall -std=c++11 main.cpp -o myProgram.exe`

Here are some important ones to recognize:

-g	Compile with debug information built in. Necessary for meaningful debugging but does bloat executable, make it much slower to run and makes it easier for others to reverse engineer.
-std=c++11	Compiles the code using the more modern 2011 version of the C++ spec (instead of the ancient one)
-O0	Capital Oh followed by zero. No optimization (default)
-O2	All reasonable code optimizations - compile time is slower, but program will run significantly faster (building release in QTCreator).
-O3	Even more optimization - these may make the executable much larger in an effort to speed things up.
-Wall	Turn on most warnings.
-o	Output file name.
-c	Create object files – do not link into application.
-S	Output an assembly file instead of compiling to machine code.
-E	Just run preprocessor and dump to console with notes about what line numbers code came from. **-E -P** skips line numbers.
-I*folder*	Look in given folder (may be relative using . or .. or absolute starting with C:…) for header files that are included.
-L*folder*	Look in given folder for library files needed.
-l*folder*	Lower-case L : library to link against.

In general, they can go in any order. If two things conflict (-O0 and -O3), last one usually wins.

Redirection:

Allow you to send program output to a file or a file's contents to a program. (See http://www.cs.princeton.edu/courses/archive/spr05/cos126/cmd-prompt.html for more details).

`myProgram.exe > out.txt`
Run myProgram.exe and whatever it prints send to the file out.txt instead of the console. (If the file exists already, overwrite it). This includes any input prompts… you have to know what needs to be typed in as input to the program and do so without the prompts!

`myProgram.exe >> out.txt`
Run myProgram.exe and whatever it prints ADD to the end of file out.txt instead of the console. (If the file does not exist, make it).

`myProgram.exe < in.txt`
Run myProgram.exe and whenever it needs input, get it from in.txt instead of the console.

`myProgram.exe 2> error.txt`
Run myProgram.exe and whatever error messages it prints send to the file error.txt instead of the console. This is handy for storing compiler error messages that g++ gives. Something like:
`g++ myBadFile.cpp -o myProgram.exe 2> error.txt`