Simple Introduction to Makefiles

Makefiles provide an "easy" way to organize compilation of your code.
This document provides only the most basic of examples. Its intent is only to demonstrate how to create and use a makefile to compile and link multiple files to create an executable file.

Assume there are three files such as:

HelloMain.cpp

```
#include "HelloFunction.h"

int main()
{
   HelloFunction();
   return 0;
}
```

HelloFunction.cpp

```
#include "HelloFunction.h"

void HelloFunction()
{
   cout << "Hello there\n";
}
```

HelloFunction.h

```
// include standard stuff

void HelloFunction();
```

From the terminal window at the command prompt, these files would normally be compiled and linked like:

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

Aside: this assumes g++ looks in the current directory for. h files. You may need -I at the end of the g++ line

And then to run the result

```
./hello
```

This is not bad when there are just 2 files. But when there are 20 or 30 it begins to get tiresome to type and is prone to error. To mitigate this you can create a makefile (literally the file is named makefile) in the same directory as your cpp and h files. **TABS and end-of-lines need to be unix style so create your makefile in linux**.

A possible makefile to compile and link the above 3 files might look something like:

makefile

```
hello: HelloMain.o HelloFunction.o
	g++ -o hello HelloMain.o HelloFunction.o

HelloMain.o: HelloMain.cpp
	g++ -c HelloMain.cpp

HelloFunction.o: HelloFunction.cpp
	g++ -c HelloFunction.cpp

clean:
	rm -rf *.o hello
```

Once the above makefile is created then (in the same directory as the files)
To compile and link only requires typing:

```
make hello
```

This will produce the executable file named hello. Which again can be run via

```
./hello
```

How does it work?

makefile	
`hello: HelloMain.o HelloFunction.o`	This says "hello" is made from HelloMain.o and HelloFunction.o
`g++ -o hello HelloMain.o HelloFunction.o`	This says take those 2 files and using g++ link them together. Note the –o hello tells g++ to name the resulting executable hello **There is a tab before the g++**
`HelloMain.o: HelloMain.cpp`	This says HelloMain.o is made from HelloMain.cpp
`g++ -c HelloMain.cpp`	This says take that file and compile it. The –c option means just compile. This defaults to creating a file named HelloMain.o **There is a tab before the g++**
`HelloFunction.o: HelloFunction.cpp`	This says HelloFunction.o is made from HelloFunction.cpp
`g++ -c HelloFunction.cpp`	This says take that file and compile it. The –c option means just compile. This defaults to creating a file named HelloFunction.o **There is a tab before the g++**
`clean:`	This says clean needs no files
`rm -rf *.o hello`	This says recursively remove all files ending in .o and the file named hello **There is a tab before the rm**

In Linux there is a "make" program installed. To run it you just type

make *[argument]*

Overlooking and simplifying things in an extreme manner:

When the make program runs, it looks in the current directory for a text file named makefile

Inside the makefile it looks for *[argument]*

If *[argument]* is made from other stuff, the make program will then go look for that other stuff in the current directory. If it finds it in the current directory it uses what is there. If it does not find it then it looks in the makefile to see if there are directions on how to make the missing stuff.

Once all the required stuff is found or made the make program then executes the "next line" after *[argument]*:

So if you type

make clean

This requires no files so the "next line" is immediately executed

rm –rf *.o hello

So when you type

make clean

All files ending in .o and the file hello will be recursively removed.

Now if you type

```
make hello
```

The make program discovers this requires HelloMain.o and HelloFunction.o to exist in the current directory.
Assuming they do, the make program then executes

```
g++ -o hello HelloMain.o HelloFunction.o
```

And you get the desired executable file named hello.

If they do not exist then the make program looks in the Makefile for how to create those .o files.
For HelloMain.o it discovers it in turn requires the file HelloMain.cpp to exist in the current directory.
Assuming it does the make program then executes

```
g++ -c HelloMain.cpp
```

which creates the needed HelloMain.o file

Likewise occurs for HelloFunction.o

Once both HelloMain.o and HelloFunction.o are successfully created then the make program returns to the line

```
hello: HelloMain.o HelloFunction.o
```

And then executes

```
g++ -o hello HelloMain.o HelloFunction.o
```

And you get the desired executable file named hello

In sum you would want to type

```
make clean
make hello
```

Each time you wanted to test changes in the source code.

How to change it for source files of a different name and/or more source files

If you happen to have just two files then it is just a search and replace
hello is the name of the desired executable
HelloMain is the prefix of one of your source files
HelloFunction is the prefix of the other source file

makefile

```
hello: HelloMain.o HelloFunction.o
	g++ -o hello HelloMain.o HelloFunction.o

HelloMain.o: HelloMain.cpp
	g++ -c HelloMain.cpp

HelloFunction.o: HelloFunction.cpp
	g++ -c HelloFunction.cpp

clean:
	rm -rf *.o hello
```

If you have more than three files then the first 2 are as above.
The third file (and any other additional cpp files) requires extra stuff added
But it is really just doing the same thing that is being done for the first 2
Assume the third file name is ThirdFile.cpp…

makefile

```
hello: HelloMain.o HelloFunction.o ThirdFile.o
	g++ -o hello HelloMain.o HelloFunction.o ThirdFile.o

HelloMain.o: HelloMain.cpp
	g++ -c HelloMain.cpp

HelloFunction.o: HelloFunction.cpp
	g++ -c HelloFunction.cpp

ThirdFile.o: ThirdFile.cpp
	g++ -c ThirdFile.cpp

clean:
	rm -rf *.o hello
```

So for four files, with the fourth file named File4.cpp, it would look like:

makefile

```
hello: HelloMain.o HelloFunction.o ThirdFile.o File4.o
	g++ -o hello HelloMain.o HelloFunction.o ThirdFile.o File4.o

HelloMain.o: HelloMain.cpp
	g++ -c HelloMain.cpp

HelloFunction.o: HelloFunction.cpp
	g++ -c HelloFunction.cpp

ThirdFile.o: ThirdFile.cpp
	g++ -c ThirdFile.cpp

File4.o: File4.cpp
	g++ -c File4.cpp

clean:
	rm -rf *.o hello
```

And so on…