





CSO Tutorial - 1

 Created	Empty
 Created by Krishna	Empty
 Tags	Empty
 Property	Empty

Some useful GCC command-line options.

This tutorial will use the program below called main.c -

```
#include<stdio.h> int main() { printf("\n CSO First tutorial \n");  
return 0; }
```

Use the command below to compile the program

```
gcc main.c
```

The above command executes the complete compilation process and outputs an executable with the name a.out.

1. Use option -o, as shown below, to specify the output file name for the executable.

```
gcc main.c -o main
```

2. Enable all warnings set through -Wall option

This option enables all the warnings in GCC

For example for the code below -

```
#include<stdio.h> int main(void) { int i; printf("\n Welcome to CS0
Tute-%d \n",i+1); return 0; }
```

```
$ gcc -Wall main.c -o main main.c: In function 'int main()':
main.c:6:10: warning: 'i' is used uninitialized in this function [-
Wuninitialized] 6 | printf("\n Welcome to CS0 Tute-%d \n",i+1); |
~~~~~^~~~~~
```

3. Produce only the pre-processor output with -E option

```
$ gcc -E main.c > main.i
```

The gcc command produces the output on stdout so you can redirect the output in any file. In our case (above), the file main.i would contain the preprocessed output.

4. Produce only the assembly code using -S option

```
gcc -S main.c > main.s
```

In this case, the file main.s would contain the assembly output.

5. Produce only the compiled code using the -C option

To produce only the compiled code (without any linking), use the -C option.

```
gcc -C main.c
```

The command above would produce a file main.o that would contain machine-level code or the compiled code.

To look at all the intermediary files generated during the compilation processes use the command below -

```
gcc -Wall -save-temps main.c -o main
```

```
$ gcc -save-temps main.c $ ls a.out main.c main.i main.o main.s
```

So we see that all the intermediate files as well as the final executable were produced in the output.

6. Produce all the intermediate files using the -save-temps function

The option -save-temps can do all the work done in examples 4,5 and 6 above. Through this option, output at all the stages of compilation is stored in the current directory. Please note that this option produces the executable also.

For example :

```
$ gcc -save-temps main.c $ ls a.out main.c main.i main.o main.s
```

8. Link with shared libraries using the -l option

The option -l can be used to link with shared libraries. For example, it can be used to link with a standard library called math.h

```
#include<stdio.h> int main(void) { int i; printf("\n Welcome to CS0  
Tute-%d \n",i+1); srand(1); printf("\n\n Displaying a random  
integer %d\n\n",rand()); return 0; }
```

```
gcc -Wall main.c -o main -lm ./main
```

The GCC command mentioned above links the code main.c with the standard shared library math.h [libm.so](#) (search for this file) to produce the final executable 'main'.

Output -

```
Welcome to CSO Tute-1 Displaying a random integer 1804289383
```

9. Print all the verbose information/additional details using -v option

The option -v can be used to provide verbose information on all the steps gcc takes while compiling a source file.

For example :

```
$ gcc -Wall -v main.c -o main Using built-in specs. COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/i686-linux-gnu/4.6/lto-wrapper Tar
get: i686-linux-gnu Configured with: ../src/configure -v --with-pkg
version='Ubuntu/Linaro 4.6.3-1ubuntu5' --with-bugurl=file:///usr/sh
are/doc/gcc-4.6/README.Bugs --enable-languages=c,c++,fortran,objc,o
bj-c++ --prefix=/usr --program-suffix=-4.6 --enable-shared --enable
-linker-build-id --with-system-zlib --libexecdir=/usr/lib --without
-included-gettext --enable-threads=posix --with-gxx-include-dir=/us
r/include/c++/4.6 --libdir=/usr/lib --enable-nls --with-sysroot=/ -
-enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-tim
e=yes --enable-gnu-unique-object --enable-plugin --enable-objc-gc -
-enable-targets=all --disable-werror --with-arch-32=i686 --with-tun
e=generic --enable-checking=release --build=i686-linux-gnu --host=i
686-linux-gnu --target=i686-linux-gnu Thread model: posix gcc versi
on 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5) ... ..
```

10. Enable the support of ISO C89 programs using -ansi option

Through -ansi option the support for ISO C89 style is enabled.

Consider the following code

```
#include<stdio.h> int main(void) { // Print the string printf("\n Welcome to CS0 Tute\n"); return 0; }
```

If the above code is compiled with -ansi option then gcc would produce an error because the C++ comments are not allowed in ISO C89 style.

Here is the output:

```
$ gcc -Wall -ansi main2.c -o main main2.c: In function 'main':  
main2.c:5:3: error: C++ style comments are not allowed in ISO C90  
// Print the string ^ main2.c:5:3: error: (this will be reported  
only once per input file)
```

11. Interpret char as unsigned char using -funsigned-char option

Through this option, the char type is treated as unsigned type.

Here is an example :

```
#include<stdio.h> int main(void) { char c = -10; // Print the string printf("\n Welcome to CS0 Tute [%d]\n", c); return 0; }
```

When the above code is compiled with funsigned-char option, here is the output :

```
$ gcc -Wall -funsigned-char main.c -o main $ ./main Welcome to CS0  
Tute [246]
```

So we see that the char was indeed treated as unsigned.

12. Interpret char as signed char using -fsigned-char option

This is the opposite of what we discussed in (12) above. Using this flag, the char variables are treated as signed.

Here is an example :

```
$ gcc -Wall -fsigned-char main.c -o main $ ./main Welcome to CS0 Tute [-10]
```

The output confirms that char was treated as signed.

13. Use compile time macros using -D option

The compiler option D can be used to define compile time macros in code.

Here is an example :

```
#include<stdio.h> int main(void) { #ifdef MY_MACRO printf("\n Macro defined \n"); #endif char c = -10; // Print the string printf("\n Welcome to CS0 Tute [%d]\n", c); return 0; }
```

The compiler option -D can be used to define the macro MY_MACRO from command line.

```
$ gcc -Wall -DMY_MACRO main.c -o main $ ./main Macro defined Welcome to CS0 Tute [-10]
```

14. Convert warnings into errors with -Werror option

Through this option, any warning that gcc could report gets converted into error.

Here is an example :

```
#include<stdio.h> int main(void) { char c; // Print the string printf("\n The Geek Stuff [%d]\n", c); return 0; }
```

The compilation of above code should generate warning related to undefined variable c and this should get converted into error by using -Werror option.

```
$ gcc -Wall -Werror main.c -o main main.c: In function 'main':  
main.c:7:4: error: 'c' is used uninitialized in this function [-Werror=uninitialized] printf("\n Welcome to CSO Tute [%d]\n", c);  
^~~~~~ cc1.exe: all warnings  
being treated as errors
```

15. Provide gcc options through a file using @option

The options for gcc can also be provided through a file. This can be done using the @ option followed by the file name containing the options. More than one option is separated by white space.

Here is an example :

```
$ cat opt_file -Wall -o main
```

The opt_file contains the options.

Now compile the code by providing opt_file along with option @.

```
$ gcc main.c @opt_file main.c: In function 'main': main.c:6:11: war  
ning: 'i' is used uninitialized in this function [-Wuninitialized]  
$ ls main main
```

The output confirms that file opt_file was parsed to get the options and the compilation was done accordingly.

Additional info:

To obtain a brief reminder of various command-line options, GCC provides a help option which displays a summary of the top-level GCC command-line options:

```
$ gcc --help
```

To display a complete list of options for `gcc` and its associated programs, such as the GNU Linker and GNU Assembler, use the help option above with the verbose (`-v`) option:

```
$ gcc -v --help
```

The complete list of options produced by this command is extremely long--you may wish to page through it using the `more` command, or redirect the output to a file for reference:

```
$ gcc -v --help 2>&1 | more
```