FREE eBook

LEARNING

gcc

Free unaffiliated eBook created from
Stack Overflow contributors.

#gcc

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gcc

It is an unofficial and free gcc ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gcc.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with gcc

Remarks

GCC (upper case) refers to the GNU Compiler Collection. This is an open source compiler suite which include compilers for C, C++, Objective C, Fortran, Ada, Go and Java. gcc (lower case) is the C compiler in the GNU Compiler Collection. Historically GCC and gcc have been used interchangeably, but efforts are being made to separate the two terms as GCC contains tools to compile more than C.

Documentation in this section will refer to gcc, the GNU C compiler. The intent is to provide a quick lookup of common actions and options. The GCC project has detailed documentation at https://gcc.gnu.org which document installation, general usage, and every command line option. Please refer to the official GCC documentation on any question not answered here. If a certain topic is unclear in the GCC documentation, please request specific examples.

Versions

Version	Release Date
7.1	2017-05-02
6.3	2016-12-21
6.2	2016-08-22
5.4	2016-06-03
6.1	2016-04-27
5.3	2015-12-04
5.2	2015-07-16
5.1	2015-04-22
4.9	2014-04-22
4.8	2013-03-22
4.7	2012-03-22
4.6	2011-03-25
4.5	2010-04-14
4.4	2009-04-21

Version	Release Date
4.3	2008-03-05
4.2	2007-05-13
4.1	2006-02-28
4.0	2005-04-20
3.4	2004-04-18
3.3	2003-05-13
3.2	2002-08-14
3.1	2002-05-15
3.0	2001-06-18
2.95	1999-07-31
2.8	1998-01-07
2.7	1995-06-16
2.6	1994-07-14
2.5	1993-10-22
2.4	1993-05-17
2.3	1992-10-31
2.2	1992-06-08
2.1	1992-03-24
2.0	1992-02-22
1.42	1992-09-20
1.41	1992-07-13
1.40	1991-06-01
1.39	1991-01-16
1.38	1990-12-21
1.37	1990-02-11

Version	Release Date
1.36	1989-09-24
1.35	1989-04-26
1.34	1989-02-23
1.33	1989-02-01
1.32	1988-12-21
1.31	1988-11-19
1.30	1988-10-13
1.29	1988-10-06
1.28	1988-09-14
1.27	1988-09-05
1.26	1988-08-18
1.25	1988-08-03
1.24	1988-07-02
1.23	1988-06-26
1.22	1988-05-22
1.21	1988-05-01
1.20	1988-04-19
1.19	1988-03-29
1.18	1988-02-04
1.17	1988-01-09
1.16	1987-12-19
1.15	1987-11-28
1.14	1987-11-06
1.13	1987-10-12
1.12	1987-10-03

Version	Release Date
1.11	1987-09-05
1.10	1987-08-22
1.9	1987-08-18
1.8	1987-08-10
1.7	1987-07-21
1.6	1987-07-02
1.5	1987-06-18
1.4	1987-06-13
1.3	1987-06-10
1.2	1987-06-01
1.1	1987-05-24
1.0	1987-05-23
0.9	1987-03-22

Examples

"Hello world!" with common command line options

For programs with a single source file, using gcc is simple.

```
/* File name is hello_world.c */
#include <stdio.h>

int main(void)
{
    int i;
    printf("Hello world!\n");
}
```

To compile the file hello_world.c from the command line:

```
gcc hello_world.c
```

gcc will then compile program and output the executable to the file a.out. If you want to name the executable, use the -o option.

```
gcc hello_world.c -o hello_world
```

The executable will then be named hello_world instead of a.out. By default, there are not that many warnings that are emitted by gcc. gcc has many warning options and it is a good idea to look through the gcc documentation to learn what is available. Using '-Wall' is a good starting point and covers many common problems.

```
gcc -Wall hello_world.c -o hello_world
```

Output:

```
hello_world.c: In function ‘main’:
hello_world.c:6:9: warning: unused variable ‘i’ [-Wunused-variable]
     int i;
         ^
```

Here we see we now get a warning that the variable 'i' was declared but not used at all in the function.

If you plan to use a debugger for testing your program, you'll need to tell gcc to include debugging information. Use the '-g' option for debugging support.

```
gcc -Wall -g hello_world.c -o hello_world
```

hello_world now has debugging information present supported by GDB. If you use a different debugger, you may need to use different debugging options so the output is formatted correctly. See the official gcc documentation for more debugging options.

By default gcc compiles code so that it is easy to debug. gcc can optimize the output so that the final executable produces the same result but has faster performance and may result in a smaller sized executable. The '-O' option enables optimization. There are several recognized qualifiers to add after the O to specify the level of optimization. Each optimization level adds or removes a set list of command line options. '-O2', '-Os', '-O0' and '-Og' are the most common optimization levels.

```
gcc -Wall -O2 hello_world.c -o hello_world
```

'-O2' is the most common optimization level for production-ready code. It provides an excellent balance between performance increase and final executable size.

```
gcc -Wall -Os hello_world.c -o hello_world
```

'-Os' is similar to '-O2', except certain optimizations that may increase execution speed by increasing the executable size are disabled. If the final executable size matters to you, try '-Os' and see if there is a noticeable size difference in the final executable.

```
gcc -Wall -g -Og hello_world.c -o -hello_world
```

Note that in the above examples with '-Os' and '-O2', the '-g' option was removed. That is because when when you start telling the compiler to optimize the code, certain lines of code may in essence no longer exist in the final executable making debugging difficult. However, there are also cases where certain errors occur only when optimizations are on. If you want to debug your application and have the compiler optimize the code, try the '-Og' option. This tells gcc to perform all optimizations that should not hamper the debugging experience.

```
gcc -Wall -g -O0 hello_world.c -o hello_world
```

'-O0' performs even less optimizations than '-Og'. This is the optimization level gcc uses by default. Use this option if you want to make sure that optimizations are disabled.

Determine gcc version

When referring to gcc's documentation, you should know which version of gcc you are running. The GCC project has a manual for each version of gcc which includes features that are implemented in that version. Use the '-v' option to determine the version of gcc you are running.

```
gcc -v
```

Example Output:

```
Using built-in specs.
COLLECT_GCC=/usr/bin/gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/5.3.1/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,objc,obj-
c++,fortran,ada,go,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-
bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-
checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-
libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-linker-hash-
style=gnu --enable-plugin --enable-initfini-array --disable-libgcj --with-default-libstdcxx-
abi=gcc4-compatible --with-isl --enable-libmpx --enable-gnu-indirect-function --with-
tune=generic --with-arch_32=i686 --build=x86_64-redhat-linux
Thread model: posix
gcc version 5.3.1 20160406 (Red Hat 5.3.1-6) (GCC)
```

In this example we see that we are running gcc version 5.3.1. You would then know to refer to the GCC 5.3 manual. It is also helpful to include your gcc version when asking questions in case you have a version specific problem.

Read Getting started with gcc online: https://riptutorial.com/gcc/topic/3193/getting-started-with-gcc

Chapter 2: Code coverage: gcov

Remarks

GCC provide some documentation of gcov here

Gcovr and Lcov can be used to help generate and summarize the coverage results

Examples

Introduction

Code coverage is a measure used to how often each source code statement and branch is executed. This measure is usually required when running a test suite to ensure that as much of the code as possible is tested by the test suite. It can also be used during profiling to determine code hot-spots and thus where optimization efforts may have the most effect.

In GCC code coverage is provided by the gcov utility. gcov works only with code compiled with gcc with particular flags. There are very few other compilers with which gcov works at all.

Compilation

Before using gcov, source code should be compiled with gcc using the two flags, `-fprofile-arcs` and `-ftest-coverage`. This tells the compiler to generate the information and extra object file code required by gcov.

```
gcc -fprofile-arcs -ftest-coverage hello.c
```

Linking should also use the `-fprofile-arcs` flag.

Generate Output

To generate the coverage information the compiled program should be executed. When creating code coverage for a test suite this execution step will normally be performed by the test suite so that the coverage shows what parts of the program the tests executes and which they do not.

```
$ a.out
```

Executing the program will cause a `.gcda` file to be generated in the same directory as the object file.

Subsequently you can call gcov with the program's source file name as an argument to produce a listing of the code with frequency of execution for each line.

```
$ gcov hello.c
```

```
File 'hello.c'
Lines executed:90.00% of 10
Creating 'hello.c.gcov'
```

The result is contained in a `.gcov` file. Here is a sample:

```
        -:    0:Source:hello.c
        -:    0:Graph:hello.gcno
        -:    0:Data:hello.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:#include <stdio.h>
        -:    2:
        -:    3:int main (void)
        1:    4:{
        1:    5:  int i;
        -:    6:
        1:    7:  i = 0;
        -:    8:
        -:    9:
        1:   10:  if (i != 0)
    #####:   11:    printf ("Goodbye!\n");
        -:   12:  else
        1:   13:    printf ("Hello\n");
        1:   14:  return 0;
        -:   15:}
```

Here you can see the line numbers and source and the number of times each line executed. If a line did not execute it is marked with `#####`.

The execution counts are cumulative. If the example program were executed again without removing the .gcda file, the count for the number of times each line in the source was executed would be added to the results of the previous run.

Read Code coverage: gcov online: https://riptutorial.com/gcc/topic/7873/code-coverage--gcov

Chapter 3: GCC Optimizations

Introduction

The GNU compiler offers various levels of optimizations for the compilation process. These optimizations are used to improve the code performance and/or code size. Compiling a code with optimizations on, typically takes longer to complete.

This command tells you what optimizations are available on your system : $gcc -Q --help=optimizations

Here is a detailed documentation of options to control optimizations:

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Examples

Differnce between codes compiled with O0 and O3

I wrote a simple C code foo.c

```
int main()
{
    int i = 0;
    int j = 0;
    for (i = 0; i < 5; i++) {
            j = i + 1;
    }
    return 0;
}
```

When compiled with -O0 i.e. by disabling all compiler optimizations

```
$ gcc -o foo.S foo.c -O0 -S
```

I got this:

```
    .file   "foo.c"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    $0, -4(%rbp)
```

```
    movl    $0, -8(%rbp)
    movl    $0, -4(%rbp)
    jmp     .L2
.L3:
    movl    -4(%rbp), %eax
    addl    $1, %eax
    movl    %eax, -8(%rbp)
    addl    $1, -4(%rbp)
.L2:
    cmpl    $4, -4(%rbp)
    jle     .L3
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (GNU) 6.2.0"
    .section        .note.GNU-stack,"",@progbits
```

GCC took all the pain to convert my code into assembly language verbatim.

But when I compiled my code with O3 i.e. with the highest level of optimizations

I got this:

```
.file   "foo.c"
.section        .text.startup,"ax",@progbits
.p2align 4,,15
.globl  main
.type   main, @function
main:
.LFB11:
    .cfi_startproc
    xorl    %eax, %eax
    ret
    .cfi_endproc
.LFE11:
    .size   main, .-main
    .ident  "GCC: (GNU) 6.2.0"
    .section        .note.GNU-stack,"",@progbits
```

GCC understood that I was just doodling and doing nothing important with the variables and the loop. So it left me a blank stub with no code.

DAYUM!

Read GCC Optimizations online: https://riptutorial.com/gcc/topic/10568/gcc-optimizations

Chapter 4: GNU C Extensions

Introduction

The GNU C compiler comes with some cool features that are not specified by the C standards. These extensions are heavily used in system software and are a great tool for performance optimization.

Examples

Attribute packed

packed is a variable attribute that is used with structures and unions in order to minimize the memory requirements.

```
#include <stdio.h>
struct foo {
    int a;
    char c;
};

struct __attribute__((__packed__))foo_packed {
    int a;
    char c;
};

int main()
{
    printf("Size of foo: %d\n", sizeof(struct foo));
    printf("Size of packed foo: %d\n", sizeof(struct foo_packed));
    return 0;
}
```

On my 64 bit Linux,

- Size of struct foo = 8 bytes
- Size of struct foo_packed = 5 bytes

packed attribute curbs the structure padding that the compiler performs to maintain memory alignment.

Read GNU C Extensions online: https://riptutorial.com/gcc/topic/10567/gnu-c-extensions

Chapter 5: Warnings

Syntax

- **gcc** [**-W***option* [**-W***option* [...]]] *src-file*

Parameters

Parameter	Details
option	It can be used to enable or disable warnings. It can make warnings into errors.
src-file	The source file to be compiled.

Remarks

It is a good practice to enable most warnings while developing a software.

Examples

Enable nearly all warnings

C source file

```
gcc -Wall -Wextra -o main main.c
```

C++ source file

```
g++ -Wall -Wextra -Wconversion -Woverloaded-virtual -o main main.cpp
```

Read Warnings online: https://riptutorial.com/gcc/topic/6501/warnings

Credits

S. No	Chapters	Contributors
1	Getting started with gcc	bevenson, Community, Dmitry Grigoryev, nachiketkulk, tversteeg
2	Code coverage: gcov	Toby
3	GCC Optimizations	nachiketkulk
4	GNU C Extensions	nachiketkulk
5	Warnings	M. Sadeq H. E.