EBook Gratis

APRENDIZAJE

gcc

Free unaffiliated eBook created from
Stack Overflow contributors.

#gcc

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con gcc

Observaciones

GCC (mayúsculas) se refiere a la colección de compiladores GNU. Este es un conjunto de compiladores de código abierto que incluye compiladores para C, C ++, Objective C, Fortran, Ada, Go y Java. gcc (minúscula) es el compilador de C en la colección de compiladores de GNU. Históricamente, GCC y gcc se han usado indistintamente, pero se están haciendo esfuerzos para separar los dos términos, ya que GCC contiene herramientas para compilar más de C.

La documentación en esta sección se referirá a gcc, el compilador de GNU C. La intención es proporcionar una búsqueda rápida de acciones y opciones comunes. El proyecto GCC tiene documentación detallada en https://gcc.gnu.org que documenta la instalación, el uso general y cada opción de línea de comando. Consulte la documentación oficial de GCC sobre cualquier pregunta que no haya respondido aquí. Si un tema determinado no está claro en la documentación de GCC, solicite ejemplos específicos.

Versiones

Versión	Fecha de lanzamiento
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

Versión	Fecha de lanzamiento
4.4	2009-04-21
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

Versión	Fecha de lanzamiento
1.37	1990-02-11
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

Versión	Fecha de lanzamiento
1.12	1987-10-03
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

"¡Hola Mundo!" con opciones de línea de comando comunes

Para programas con un solo archivo fuente, usar gcc es simple.

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

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

Para compilar el archivo hello_world.c desde la línea de comando:

```
gcc hello_world.c
```

gcc luego compilará el programa y enviará el archivo ejecutable al archivo a.out. Si desea asignar

un nombre al ejecutable, use la opción -o.

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

El ejecutable se llamará hello_world en lugar de a.out. Por defecto, no hay muchas advertencias emitidas por gcc. gcc tiene muchas opciones de advertencia y es una buena idea revisar la documentación de gcc para saber qué hay disponible. Usar '-Wall' es un buen punto de partida y cubre muchos problemas comunes.

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

Salida:

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

Aquí vemos que ahora recibimos una advertencia de que la variable 'i' fue declarada pero no utilizada en absoluto en la función.

Si planea usar un depurador para probar su programa, deberá indicar a gcc que incluya información de depuración. Use la opción '-g' para el soporte de depuración.

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

hello_world ahora tiene información de depuración presente soportada por GDB. Si usa un depurador diferente, es posible que deba usar diferentes opciones de depuración para que la salida tenga el formato correcto. Consulte la documentación oficial de gcc para obtener más opciones de depuración.

Por defecto, gcc compila el código para que sea fácil de depurar. gcc puede optimizar la salida para que el ejecutable final produzca el mismo resultado pero tenga un rendimiento más rápido y pueda resultar en un ejecutable de menor tamaño. La opción '-O' habilita la optimización. Hay varios calificadores reconocidos para agregar después de la O para especificar el nivel de optimización. Cada nivel de optimización agrega o elimina una lista de opciones de línea de comandos. '-O2', '-Os', '-O0' y '-Og' son los niveles de optimización más comunes.

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

'-O2' es el nivel de optimización más común para el código listo para producción. Proporciona un excelente equilibrio entre el aumento de rendimiento y el tamaño ejecutable final.

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

'-Os' es similar a '-O2', excepto que ciertas optimizaciones que pueden aumentar la velocidad de ejecución al aumentar el tamaño del ejecutable están deshabilitadas. Si el tamaño del ejecutable

final es importante para usted, intente '-Os' y vea si hay una diferencia de tamaño notable en el ejecutable final.

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

Tenga en cuenta que en los ejemplos anteriores con '-Os' y '-O2', se eliminó la opción '-g'. Esto se debe a que cuando empiezas a decirle al compilador que optimice el código, es posible que ciertas líneas de código ya no existan en el ejecutable final, lo que dificulta la depuración. Sin embargo, también hay casos en que ciertos errores ocurren solo cuando las optimizaciones están activadas. Si desea depurar su aplicación y hacer que el compilador optimice el código, pruebe la opción '-Og'. Esto le indica a gcc que realice todas las optimizaciones que no deben obstaculizar la experiencia de depuración.

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

'-O0' realiza incluso menos optimizaciones que '-Og'. Este es el nivel de optimización que usa gcc por defecto. Utilice esta opción si desea asegurarse de que las optimizaciones están deshabilitadas.

Determinar la versión gcc

Cuando consulte la documentación de gcc, debe saber qué versión de gcc está ejecutando. El proyecto GCC tiene un manual para cada versión de gcc que incluye características que se implementan en esa versión. Use la opción '-v' para determinar la versión de gcc que está ejecutando.

```
gcc -v
```

Ejemplo de salida:

```
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)
```

En este ejemplo vemos que estamos ejecutando gcc versión 5.3.1. Entonces deberías saber consultar el manual de GCC 5.3. También es útil incluir su versión de gcc cuando haga preguntas en caso de que tenga un problema de versión específica.

Lea Empezando con gcc en línea: https://riptutorial.com/es/gcc/topic/3193/empezando-con-gcc

Capítulo 2: Advertencias

Sintaxis

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

Parámetros

Parámetro	Detalles
opción	Se puede utilizar para habilitar o deshabilitar las advertencias. Puede hacer advertencias en errores.
archivo src	El archivo fuente a compilar.

Observaciones

Es una buena práctica habilitar la mayoría de las advertencias al desarrollar un software.

Examples

Habilitar casi todas las advertencias

Archivo fuente C

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

Archivo fuente de C ++

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

Lea Advertencias en línea: https://riptutorial.com/es/gcc/topic/6501/advertencias

Capítulo 3: Cobertura del código: gcov

Observaciones

GCC proporciona alguna documentación de gcov aquí

Gcovr y Lcov se pueden usar para ayudar a generar y resumir los resultados de la cobertura

Examples

Introducción

La cobertura de código es una medida que se utiliza para determinar la frecuencia con la que se ejecuta cada instrucción de código fuente y rama. Esta medida suele ser necesaria cuando se ejecuta un conjunto de pruebas para garantizar que la mayor parte del código sea probado por el conjunto de pruebas. También se puede utilizar durante el perfilado para determinar los puntos críticos del código y, por lo tanto, donde los esfuerzos de optimización pueden tener el mayor efecto.

En el código GCC la cobertura es proporcionada por la utilidad gcov. gcov funciona solo con el código compilado con gcc con indicadores particulares. Hay muy pocos compiladores con los que funciona gcov.

Compilacion

Antes de usar gcov, el código fuente debe compilarse con gcc utilizando los dos indicadores, `-fprofile-arcs` y `-ftest-coverage` . Esto le dice al compilador que genere la información y el código de archivo de objeto adicional requerido por gcov.

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

La vinculación también debe utilizar el `-fprofile-arcs` .

Generar salida

Para generar la información de cobertura se debe ejecutar el programa compilado. Cuando se crea una cobertura de código para un conjunto de pruebas, este paso de ejecución normalmente lo realizará el conjunto de pruebas para que la cobertura muestre qué partes del programa ejecutan las pruebas y cuáles no.

```
$ a.out
```

La ejecución del programa hará que se `.gcda` un archivo `.gcda` en el mismo directorio que el archivo objeto.

Posteriormente, puede llamar a gcov con el nombre del archivo fuente del programa como un argumento para producir una lista del código con la frecuencia de ejecución para cada línea.

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

El resultado está contenido en un archivo `.gcov` . Aquí hay una muestra:

```
        -:    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:}
```

Aquí puede ver los números de línea y la fuente y el número de veces que se ejecutó cada línea. Si una línea no se ejecutó, se marca con `#####` .

Los recuentos de ejecución son acumulativos. Si el programa de ejemplo se ejecutó de nuevo sin eliminar el archivo .gcda, el recuento del número de veces que se ejecutó cada línea en la fuente se agregaría a los resultados de la ejecución anterior.

Lea Cobertura del código: gcov en línea: https://riptutorial.com/es/gcc/topic/7873/cobertura-del-codigo--gcov

Capítulo 4: Extensiones GNU C

Introducción

El compilador GNU C viene con algunas características geniales que no están especificadas por los estándares C. Estas extensiones son muy utilizadas en el software del sistema y son una gran herramienta para la optimización del rendimiento.

Examples

Atributo embalado

empaquetado es un atributo variable que se utiliza con estructuras y uniones para minimizar los requisitos de memoria.

```
#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;
}
```

En mi Linux de 64 bits,

- Tamaño de la estructura foo = 8 bytes
- Tamaño de la estructura foo_packed = 5 bytes

El atributo *empaquetado* frena el relleno de la estructura que realiza el compilador para mantener la alineación de la memoria.

Lea Extensiones GNU C en línea: https://riptutorial.com/es/gcc/topic/10567/extensiones-gnu-c

Capítulo 5: Optimizaciones GCC

Introducción

El compilador GNU ofrece varios niveles de optimizaciones para el proceso de compilación. Estas optimizaciones se utilizan para mejorar el rendimiento del código y / o el tamaño del código. La compilación de un código con optimizaciones activadas, por lo general toma más tiempo en completarse.

Este comando le dice qué optimizaciones están disponibles en su sistema: $ gcc -Q --help = optimizaciones

Aquí hay una documentación detallada de las opciones para controlar las optimizaciones:

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

Examples

Diferencia entre códigos compilados con O0 y O3.

Escribí un código C simple foo.c

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

Cuando se compila con -O0, es decir, deshabilita todas las optimizaciones del compilador

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

Tengo esto:

```
    .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 se tomó la molestia de convertir mi código en lenguaje ensamblador literalmente.

Pero cuando compilé mi código con O3, es decir, con el nivel más alto de optimizaciones

Tengo esto:

```
.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 entendió que solo estaba garabateando y sin hacer nada importante con las variables y el bucle. Así que me dejó un trozo en blanco sin código.

DAYUM!

Lea Optimizaciones GCC en línea: https://riptutorial.com/es/gcc/topic/10568/optimizaciones-gcc

Creditos

S. No	Capítulos	Contributors
1	Empezando con gcc	bevenson, Community, Dmitry Grigoryev, nachiketkulk, tversteeg
2	Advertencias	M. Sadeq H. E.
3	Cobertura del código: gcov	Toby
4	Extensiones GNU C	nachiketkulk
5	Optimizaciones GCC	nachiketkulk