Бесплатная электронная книга

УЧУСЬ

gcc

Free unaffiliated eBook created from
Stack Overflow contributors.

#gcc

Около

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

глава 1: Начало работы с gcc

замечания

GCC (верхний регистр) относится к сборнику компиляторов GNU. Это компилятор с открытым исходным кодом, который включает компиляторы для C, C ++, Objective C, Fortran, Ada, Go и Java. gcc (нижний регистр) является компилятором C в сборнике компиляторов GNU. Исторически GCC и gcc использовались взаимозаменяемо, но предпринимаются усилия по разделению этих двух терминов, поскольку GCC содержит инструменты для компиляции более C.

Документация в этом разделе будет ссылаться на gcc, компилятор GNU C. Цель состоит в том, чтобы обеспечить быстрый поиск общих действий и вариантов. В проекте GCC имеется подробная документация на https://gcc.gnu.org, в которой устанавливаются документы, общее использование и каждая опция командной строки. Пожалуйста, обратитесь к официальной документации GCC по любому вопросу, на который не ответил. Если определенная тема неясна в документации GCC, просьба запросить конкретные примеры.

Версии

Версия	Дата выхода
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

Версия	Дата выхода
4,6	2011-03-25
4.5	2010-04-14
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

Версия	Дата выхода
1,39	1991-01-16
1,38	1990-12-21
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

Версия	Дата выхода
1,14	1987-11-06
1,13	1987-10-12
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

"Привет, мир!" с общими параметрами командной строки

Для программ с одним исходным файлом использование gcc прост.

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

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

Чтобы скомпилировать файл hello_world.c из командной строки:

```
gcc hello_world.c
```

Затем gcc скомпилирует программу и выдает исполняемый файл в файл a.out. Если вы хотите назвать исполняемый файл, используйте опцию -o.

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

Затем исполняемый файл будет называться hello_world вместо a.out. По умолчанию не так много предупреждений, которые испускаются gcc. В gcc есть много вариантов предупреждения, и неплохо изучить документацию gcc, чтобы узнать, что доступно. Использование «-Wall» является хорошей отправной точкой и охватывает многие распространенные проблемы.

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

Выход:

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

Здесь мы видим, что теперь мы получаем предупреждение о том, что переменная 'i' была объявлена, но не используется вообще в функции.

Если вы планируете использовать отладчик для тестирования вашей программы, вам нужно указать gcc включить информацию об отладке. Используйте параметр -g для поддержки отладки.

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

Теперь у hello_world есть отладочная информация, поддерживаемая GDB. Если вы используете другой отладчик, вам может потребоваться использовать различные параметры отладки, чтобы выход был отформатирован правильно. См. Официальную документацию gcc для получения дополнительных параметров отладки.

По умолчанию gcc компилирует код так, что его легко отлаживать. gcc может оптимизировать вывод, чтобы конечный исполняемый файл выдавал тот же результат, но имел более высокую производительность и может привести к выполнению исполняемого файла меньшего размера. Опция '-O' обеспечивает оптимизацию. Есть несколько признанных квалификаторов, которые нужно добавить после O, чтобы указать уровень оптимизации. Каждый уровень оптимизации добавляет или удаляет заданный список параметров командной строки. '-O2', '-Os', '-O0' и '-Og' являются наиболее распространенными уровнями оптимизации.

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

«-O2» является наиболее распространенным уровнем оптимизации для готового к выпуску кода. Он обеспечивает отличный баланс между увеличением производительности и окончательным размером исполняемого файла.

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

'-O' аналогичен '-O2', за исключением некоторых оптимизаций, которые могут увеличить скорость выполнения путем увеличения размера исполняемого файла, отключены. Если для вас имеет значение конечный размер исполняемого файла, попробуйте «-O» и посмотрите, есть ли заметная разница в размере окончательного исполняемого файла.

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

Обратите внимание, что в приведенных выше примерах с '-O' и '-O2' опция '-g' была удалена. Это связано с тем, что когда вы начинаете сообщать компилятору об оптимизации кода, некоторые строки кода, по существу, больше не могут существовать в конечном исполняемом файле, что затрудняет отладку. Однако есть также случаи, когда определенные ошибки возникают только при включенной оптимизации. Если вы хотите отлаживать свое приложение и оптимизировать компилятор кода, попробуйте параметр «-Og». Это говорит gcc о выполнении всех оптимизаций, которые не должны мешать работе отладки.

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

'-O0' выполняет еще меньше оптимизаций, чем '-Og'. Это уровень оптимизации, используемый gcc по умолчанию. Используйте этот параметр, если вы хотите, чтобы оптимизация была отключена.

Определить версию gcc

Обращаясь к документации gcc, вы должны знать, какую версию gcc вы используете. В проекте GCC есть руководство для каждой версии gcc, которое включает функции, реализованные в этой версии. Используйте опцию '-v', чтобы определить версию gcc, в которой вы работаете.

```
gcc -v
```

Пример:

```
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)
```

В этом примере мы видим, что мы запускаем gcc версии 5.3.1. Затем вы знаете, что ссылаетесь на руководство GCC 5.3. Также полезно включить вашу версию gcc, задавая вопросы, если у вас есть проблема с конкретной версией.

Прочитайте Начало работы с gcc онлайн: https://riptutorial.com/ru/gcc/topic/3193/начало-работы-с-gcc

глава 2: Оптимизация GCC

Вступление

Компилятор GNU предлагает различные уровни оптимизации для процесса компиляции. Эти оптимизации используются для улучшения производительности кода и / или размера кода. Компиляция кода с оптимизацией, как правило, занимает больше времени.

Эта команда сообщает вам, какие оптимизации доступны в вашей системе: $ gcc -Q --help = optimizations

Ниже приведена подробная документация по параметрам управления оптимизацией:

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

Examples

Разница между кодами, скомпонованными с O0 и O3

Я написал простой код C foo.c

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

При компиляции с -O0, т.е. отключением всех оптимизаций компилятора

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

Я получил это:

```
    .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 взял на себя всю боль, чтобы преобразовать мой код в язык ассемблера дословно.

Но когда я скомпилировал свой код с O3, то есть с самым высоким уровнем оптимизации

Я получил это:

```
.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 понял, что я просто рисовал и ничего не делал с переменными и циклом. Поэтому он оставил мне пустой заглушку без кода.

DAYUM!

Прочитайте Оптимизация GCC онлайн: https://riptutorial.com/ru/gcc/topic/10568/оптимизация-gcc

глава 3: Покрытие кода: gcov

замечания

GCC предоставляет некоторую документацию по gcov здесь

Gcovr и Lcov могут использоваться для создания и обобщения результатов покрытия

Examples

Вступление

Покрытие кода - это мера, используемая для того, как часто выполняется каждый оператор исходного кода и ветвь. Эта мера обычно требуется при запуске набора тестов для обеспечения того, чтобы как можно больше кода проверялся набором тестов. Его также можно использовать во время профилирования для определения кодовых горячих точек и, следовательно, усилия по оптимизации могут оказать наибольшее влияние.

В GCC покрытие кода обеспечивается утилитой gcov. gcov работает только с кодом, скомпилированным с gcc с определенными флагами. Очень мало других компиляторов, с которыми gcov работает вообще.

компиляция

Перед использованием gcov исходный код должен быть скомпилирован с помощью gcc с использованием двух флагов, `-fprofile-arcs` и `-ftest-coverage` . Это говорит компилятору генерировать информацию и код дополнительного объектного файла, требуемый gcov.

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

Связывание должно также использовать флаг `-fprofile-arcs` .

Создать выход

Чтобы создать информацию о покрытии, необходимо выполнить скомпилированную программу. При создании покрытия кода для набора тестов этот шаг выполнения обычно выполняется тестовым пакетом, чтобы охват показывал, какие части программы выполняются, а какие - нет.

```
$ a.out
```

Выполнение программы приведет к `.gcda` файла `.gcda` в том же каталоге, что и файл

объекта.

Впоследствии вы можете вызвать gcov с именем исходного файла программы в качестве аргумента для создания списка кода с частотой выполнения для каждой строки.

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

Результат содержится в файле `.gcov` . Вот пример:

```
        -:    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:}
```

Здесь вы можете увидеть номера строк и источник, а также количество раз, когда каждая строка выполнена. Если строка не выполнялась, она помечена знаком `#####` .

Счет выполнения является кумулятивным. Если примерная программа была выполнена снова без удаления файла .gcda, счетчик количества попыток каждой строки в источнике будет добавлен к результатам предыдущего запуска.

Прочитайте Покрытие кода: gcov онлайн: https://riptutorial.com/ru/gcc/topic/7873/покрытие-кода--gcov

глава 4: Предупреждения

Синтаксис

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

параметры

параметр	подробности
вариант	Его можно использовать для включения или отключения предупреждений. Он может вносить предупреждения в ошибки.
ГРЦ-файл	Исходный файл для компиляции.

замечания

Хорошей практикой является использование большинства предупреждений при разработке программного обеспечения.

Examples

Включить почти все предупреждения

Исходный файл C

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

Исходный файл C ++

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

Прочитайте Предупреждения онлайн: https://riptutorial.com/ru/gcc/topic/6501/предупреждения

глава 5: Расширения GNU C

Вступление

Компилятор GNU C поставляется с некоторыми классными функциями, которые не определены стандартами C. Эти расширения широко используются в системном программном обеспечении и являются отличным инструментом для оптимизации производительности.

Examples

Атрибут упакован

упакованный - это переменный атрибут, который используется со структурами и объединениями, чтобы минимизировать требования к памяти.

```
#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;
}
```

На моем 64-битном Linux,

- Размер struct foo = 8 байт
- Размер struct foo_packed = 5 байт

упакованный атрибут препятствует заполнению структуры, которую выполняет компилятор для поддержания выравнивания памяти.

Прочитайте Расширения GNU C онлайн: https://riptutorial.com/ru/gcc/topic/10567/расширения-gnu-c

кредиты

S. No	Главы	Contributors
1	Начало работы с gcc	bevenson, Community, Dmitry Grigoryev, nachiketkulk, tversteeg
2	Оптимизация GCC	nachiketkulk
3	Покрытие кода: gcov	Toby
4	Предупреждения	M. Sadeq H. E.
5	Расширения GNU C	nachiketkulk