<https://blog.csdn.net/xiaqunfeng123/article/details/51301749>

gcc的基本用法

命令格式：gcc [选项] [文件名]

编译的四个阶段：  
-E：仅执行编译预处理；   
-c：仅执行编译操作，不进行连接操作；  
-S：将C代码转换为汇编代码；   
-o：指定生成的输出文件。

–c是使用GNU汇编器将源文件转化为目标代码之后就结束，在这种情况下,只调用了C编译器（ccl）和汇编器（as），而连接器(ld)并没有被执行，所以输出的目标文件不会包含作为Linux程序在被装载和执行时所必须的包含信息，但它可以在以后被连接到一个程序

-c表示只编译(compile)，而不连接成为可执行文件。生成同名字的 .o 目标文件。通常用于编译不包含主程序的子程序文件。  
gcc -c hello.c  
生成：hello.o

-o选项用于说明输出(output)文件名，gcc将生成一个目标(object)文件xx。  
gcc hello.c -o xqf  
或者：gcc -o xqf hello.c（顺序可以调换）  
输出：xqf 为程序可执行文件

-g 选项产生符号调试工具（GNU的gdb）所必要的符号信息，插入到生成的二进制代码中。表示编译DEBUG版本。  
想要对源代码进行调试，就必须加入这个选项。当然，会增加可执行文件的大小。  
gcc study.c -o xqf  
gcc -g study.c -o xqf\_g  
结果如下：（确实加了 -g 可执行文件后变大了一点）  
-rwxr-xr-x 1 root root 12393 Apr 19 21:39 xqf\_g  
-rwxr-xr-x 1 root root 11817 Apr 19 20:48 xqf

gcc 在产生调试符号时，同样采用了分级的思路，开发人员可以通过在 -g 选项后附加数字1、2、3指定在代码中加入调试信息的多少。默认的级别是2（-g2），此时产生的调试信息包括：扩展的符号表、行号、局部或外部变量信息。  
级别3（-g3）包含级别2中的所有调试信息以及源代码中定义的宏。  
级别1（-g1）不包含局部变量和与行号有关的调试信息，因此只能够用于回溯跟踪和堆栈转储。  
回溯追踪：指的是监视程序在运行过程中函数调用历史。  
堆栈转储：则是一种以原始的十六进制格式保存程序执行环境的方法。

-pedantic 选项：当gcc在编译不符合ANSI/ISO C 语言标准的源代码时，将产生相应的警告信息

**[objc]** [view plain](https://blog.csdn.net/xiaqunfeng123/article/details/51301749) [copy](https://blog.csdn.net/xiaqunfeng123/article/details/51301749)

1. //study.c
2. #include <stdio.h>
4. **int** main()
5. {
6. **long** **long** **int** var = 1;
7. printf("hello world!\n");
8. **return** 0;
9. }

gcc -pedantic -o mm study.c   
study.c: In function ‘main’:  
study.c:5: warning: ISO C90 does not support ‘long long’

-Wall选项：使gcc产生尽可能多的警告信息，警告信息很有可能是错误的来源，特别是隐式编程错误，所以尽量保持0 warning。  
用上面的代码：study.c，编译如下  
gcc -Wall -o he study.c   
study.c: In function ‘main’:  
study.c:5: warning: unused variable ‘var’

-Werror 选项：要求gcc将所有的警告当作错误进行处理。  
同样是上面的程序：study.c  
gcc -Werror -o haha study.c  
竟然没有错误！！   
改一下study.c  
#include <stdio.h>

**[objc]** [view plain](https://blog.csdn.net/xiaqunfeng123/article/details/51301749) [copy](https://blog.csdn.net/xiaqunfeng123/article/details/51301749)

1. **void** main()
2. {
3. **long** **long** **int** var = 1;
4. printf("hello world!\n");
5. //return 0;
6. }

再编译：  
gcc -Werror -o haha study.c  
cc1: warnings being treated as errors  
study.c: In function ‘main’:  
study.c:4: error: return type of ‘main’ is not ‘int’

gcc -Wall -o hehe study.c  
study.c:3: warning: return type of ‘main’ is not ‘int’  
study.c: In function ‘main’:  
study.c:5: warning: unused variable ‘var’

所以说：并不是所有的warning都变成 error。具体的，后面再深究。

-fPIC选项。PIC指Position Independent Code。共享库要求有此选项，以便实现动态连接(dynamic linking)。

-I 选项（大写的 i）：向头文件搜索目录中添加新的目录。  
1、用＃include"file"的时候,gcc/g++会先在当前目录查找你所制定的头文件,如  
果没有找到,他回到缺省的头文件目录找。  
如果使用-I制定了目录,他会先在你所制定的目录查找,然后再按常规的顺序去找.  
2、用＃include<file>,gcc/g++会到-I制定的目录查找,查找不到,然后将到系统的缺  
省的头文件目录查找  
例如：  
gcc –I /usr/dev/mysql/include test.c –o test.o

-l选项（小写的 l）说明库文件的名字。如果库文件为 libtest.so， 则选项为： -ltest

-L选项说明库文件所在的路径。  
例如：-L.（“.”表示当前路径）。  
      -L/usr/lib （“/usr/lib” 为路径。注：这里的路径是绝对路径）  
如果没有提供 -L选项，gcc 将在默认库文件路径下搜索

-shared选项指定生成动态连接库，不用该标志外部程序无法连接。相当于一个可执行文件， 生成 .so 文件

-static 选项，强制使用静态链接库，生成 .a 文件。因为gcc在链接时优先选择动态链接库，只有当动态链接库不存在时才使用静态链接库。加上该选项可强制使用静态链接库。  
.so 和 .a 的区别：运行时动态加载，编译时静态加载  
具体的例子在文章：linux so文件生成与链接中有讲。

多个文件一起编译：  
文件：test\_a.c  test\_b.c  
两种编译方法：  
1、一起编译  
gcc test\_a.c test\_b.c -o test

2、分别编译各个源文件，之后对编译后输出的目标文件链接  
gcc -c test\_a.c  
gcc -c test\_b.c  
gcc -o test\_a.o test\_b.o -o test

比较：第一中方法编译时需要所有文件重新编译；第二种植重新编译修改的文件，未修改的不用重新编译。