gcc 参数详解

gcc and g++分别是 gnu 的 c & c++编译器 gcc/g++在执行编译工作的时候，总共需要 4 步

1.预处理,生成.i 的文件[预处理器 cpp]
2.将预处理后的文件不转换成汇编语言,生成文件.s[编译器 egcs]
3.有汇编变为目标代码(机器代码)生成.o 的文件[汇编器 as]
4.连接目标代码,生成可执行程序[链接器 ld]
[参数详解]
-x language filename
　　设定文件所使用的语言,使后缀名无效,对以后的多个有效.也就是根据约定 C 语言的后
缀名称是.c 的，而 C++的后缀名是.C 或者.cpp,如果你很个性，决定你的 C 代码文件的后缀
名是.pig 哈哈，那你就要用这个参数,这个参数对他后面的文件名都起作用，除非到了
下一个参数的使用。
　　可以使用的参数吗有下面的这些
　　`c', `objective-c', `c-header', `c++', `cpp-output', `assembler', and `a
ssembler-with-cpp'.
　　看到英文，应该可以理解的。
　　例子用法:
　　gcc -x c hello.pig

-x none filename
　　关掉上一个选项，也就是让 gcc 根据文件名后缀，自动识别文件类型
　　例子用法:
　　gcc -x c hello.pig -x none hello2.c

-c
　　只激活预处理,编译,和汇编,也就是他只把程序做成 obj 文件
　　例子用法:
　　gcc -c hello.c
　　他将生成.o 的 obj 文件
-S
　　只激活预处理和编译，就是指把文件编译成为汇编代码。
　　例子用法
　　gcc -S hello.c
　　他将生成.s 的汇编代码，你可以用文本编辑器察看
-E
　　只激活预处理,这个不生成文件,你需要把它重定向到一个输出文件里面.

例子用法:
gcc -E hello.c > pianoapan.txt
gcc -E hello.c | more
慢慢看吧,一个 hello word 也要与处理成 800 行的代码

-o

制定目标名称,缺省的时候,gcc 编译出来的文件是 a.out,很难听,如果你和我有同感
, 改掉它,哈哈
例子用法
gcc -o hello.exe hello.c (哦,windows 用习惯了)
gcc -o hello.asm -S hello.c

-pipe

使用管道代替编译中临时文件,在使用非 gnu 汇编工具的时候,可能有些问题
gcc -pipe -o hello.exe hello.c

-ansi

关闭 gnu c 中与 ansi c 不兼容的特性,激活 ansi c 的专有特性(包括禁止一些 asm inl
ine typeof 关键字,以及 UNIX,vax 等预处理宏,

-fno-asm

此选项实现 ansi 选项的功能的一部分, 它禁止将 asm,inline 和 typeof 用作关键字。

-fno-strict-prototype

只对 g++起作用,使用这个选项,g++将对不带参数的函数,都认为是没有显式的对参数
的个数和类型说明,而不是没有参数.
而 gcc 无论是否使用这个参数,都将对没有带参数的函数,认为城没有显式说明的类型

-fthis-is-varialble

就是向传统 c++看齐,可以使用 this 当一般变量使用.

-fcond-mismatch

允许条件表达式的第二和第三参数类型不匹配,表达式的值将为 void 类型

-funsigned-char
-fno-signed-char
-fsigned-char
-fno-unsigned-char

这四个参数是对 char 类型进行设置,决定将 char 类型设置成 unsigned char(前两个参
数)或者 signed char(后两个参数)

-include file

包含某个代码,简单来说,就是便以某个文件,需要另一个文件的时候,就可以用它设
定,功能就相当于在代码中使用#include<filename>

例子用法:

gcc hello.c -include /root/pianopan.h

-imacros file

将 file 文件的宏,扩展到 gcc/g++的输入文件,宏定义本身并不出现在输入文件中

-Dmacro

相当于 C 语言中的#define macro

-Dmacro=defn

相当于 C 语言中的#define macro=defn

-Umacro

相当于 C 语言中的#undef macro

-undef

取消对任何非标准宏的定义

-Idir

在你是用#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

回先在你所制定的目录查找,然后再按常规的顺序去找.

对于#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

-I-

就是取消前一个参数的功能,所以一般在-Idir 之后使用

-idirafter dir

在-I 的目录里面查找失败,讲到这个目录里面查找.

-iprefix prefix

-iwithprefix dir

一般一起使用,当-I 的目录查找失败,会到 prefix+dir 下查找

-nostdinc

使编译器不再系统缺省的头文件目录里面找头文件,一般和-I 联合使用,明

确限定头
文件的位置

-nostdin C++
规定不在 g++指定的标准路经中搜索,但仍在其他路径中搜索,.此选项在创 libg++库
使用

-C
在预处理的时候,不删除注释信息,一般和-E 使用,有时候分析程序，用这个很方便的

-M
生成文件关联的信息。包含目标文件所依赖的所有源代码你可以用 gcc -M hello.c
来测试一下，很简单。

-MM
和上面的那个一样，但是它将忽略由#include<file>造成的依赖关系。

-MD
和-M 相同，但是输出将导入到.d 的文件里面

-MMD
和-MM 相同，但是输出将导入到.d 的文件里面

-Wa,option
此选项传递 option 给汇编程序;如果 option 中间有逗号,就将 option 分成多个选项,然
后传递给会汇编程序

-Wl.option
此选项传递 option 给连接程序;如果 option 中间有逗号,就将 option 分成多个选项,然
后传递给会连接程序.

-llibrary
制定编译的时候使用的库
例子用法
gcc -lcurses hello.c
使用 ncurses 库编译程序

-Ldir

制定编译的时候，搜索库的路径。比如你自己的库，可以用它制定目录，不然

编译器将只在标准库的目录找。这个 dir 就是目录的名称。

-O0
-O1
-O2
-O3

编译器的优化选项的 4 个级别，-O0 表示没有优化,-O1 为缺省值，-O3 优化级别最高

-g

只是编译器，在编译的时候，产生调试信息。

-gstabs

此选项以 stabs 格式声称调试信息,但是不包括 gdb 调试信息.

-gstabs+

此选项以 stabs 格式声称调试信息,并且包含仅供 gdb 使用的额外调试信息.

-ggdb

此选项将尽可能的生成 gdb 的可以使用的调试信息.

-static

此选项将禁止使用动态库，所以，编译出来的东西，一般都很大，也不需要什么
动态连接库，就可以运行.

-share

此选项将尽量使用动态库，所以生成文件比较小，但是需要系统由动态库.

-traditional

试图让编译器支持传统的 C 语言特性

运行 gcc/egcs
**********运行 gcc/egcs**********

GCC 是 GNU 的 C 和 C++ 编译器。实际上，GCC 能够编译三种语言：C、C++ 和 O
bject C（C 语言的一种面向对象扩展）。利用 gcc 命令可同时编译并连接 C 和 C++
源程序。

如果你有两个或少数几个 C 源文件，也可以方便地利用 GCC 编译、连接并生成可
执行文件。例如，假设你有两个源文件 main.c 和 factorial.c 两个源文件，现在要编
译生成一个计算阶乘的程序。

代码:

清单 factorial.c

```
int factorial (int n)
{
    if (n <= 1)
      return 1;
    else
      return factorial (n - 1) * n;
}
```

清单 main.c

```
#include  <stdio.h>
#include  <unistd.h>
int factorial (int n);
int main (int argc, char **argv)
{
    int n;
    if (argc < 2)
    {
        printf ("Usage: %s n\n", argv [0]);
        return -1;
    }
    else
    {
      n = atoi (argv[1]);
      printf ("Factorial of %d is %d.\n", n, factorial (n));
     }
    return 0;
}
```

利用如下的命令可编译生成可执行文件，并执行程序：

```
$ gcc -o factorial main.c factorial.c
$ ./factorial 5
Factorial of 5 is 120.
```

GCC 可同时用来编译 C 程序和 C++ 程序。一般来说，C 编译器通过源文件的后缀名来判断是 C 程序还是 C++ 程序。在 Linux 中，C 源文件的后缀名为 .c，而 C++ 源文件的后缀名为 .C 或 .cpp。但是，gcc 命令只能编译 C++ 源文件，而不能自动和 C++ 程序使用的库连接。因此，通常使用 g++ 命令来完成 C++ 程序的编译和连

接，该程
序会自动调用 gcc 实现编译。假设我们有一个如下的 C++ 源文件（hello.C）：
```
#include <iostream>
void main (void)
{
    cout << "Hello, world!" << endl;
}
```
则可以如下调用 g++ 命令编译、连接并生成可执行文件：
```
$ g++ -o hello hello.C
$ ./hello
Hello, world!
```
**********gcc/egcs 的主要选项*********
gcc 命令的常用选项
选项 解释
-ansi 只支持 ANSI 标准的 C 语法。这一选项将禁止 GNU C 的某些特色，例如 asm 或 typeof 关键词。
-c 只编译并生成目标文件。
-DMACRO 以字符串“1”定义 MACRO 宏。
-DMACRO=DEFN 以字符串“DEFN”定义 MACRO 宏。
-E 只运行 C 预编译器。
-g 生成调试信息。GNU 调试器可利用该信息。
-IDIRECTORY 指定额外的头文件搜索路径 DIRECTORY。
-LDIRECTORY 指定额外的函数库搜索路径 DIRECTORY。
-lLIBRARY 连接时搜索指定的函数库 LIBRARY。
-m486 针对 486 进行代码优化。
-o FILE 生成指定的输出文件。用在生成可执行文件时。
-O0 不进行优化处理。
-O 或 -O1 优化生成代码。
-O2 进一步优化。
-O3 比 -O2 更进一步优化，包括 inline 函数。
-shared 生成共享目标文件。通常用在建立共享库时。
-static 禁止使用共享连接。
-UMACRO 取消对 MACRO 宏的定义。
-w 不生成任何警告信息。
-Wall 生成所有警告信息。