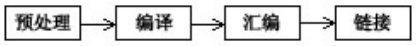
Linux 下 gcc 与 g++用法以及编写 makefile

(本文主要以 gcc 为演示，如果是 C++程序直接将给 gcc 改为 g++即可)

1. gcc 与 g++编译流程：
   1) 编译流程：



   - 链接：生成链接文件。

   2) 预处理：生成.i 的预处理文件。
      - 只激活预处理，这个不生成文件，需要把它重定向一个输出文件。
      - 演示：

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# gcc -E hello.c -o hello.i
[root@localhost demo]# ls
hello.c  hello.i
```

   3) 编译：生成.s 的编译文件。
      - 只激活预处理和编译，把文件编译成汇编代码。
      - 演示：

```
[root@localhost demo]# ls
hello.c  hello.i
[root@localhost demo]# gcc -S hello.i -o hello.s
[root@localhost demo]# ls
hello.c  hello.i  hello.s
```

   4) 汇编：生成.o 的汇编文件。
      - 只激活预处理，编译和汇编，把程序做成 obj 文件。
      - 演示：

```
[root@localhost demo]# ls
hello.c  hello.i  hello.s
[root@localhost demo]# gcc -c hello.s -o hello.o
[root@localhost demo]# ls
hello.c  hello.i  hello.o  hello.s
```

   5) 链接：
      - gcc hello.o -o hello
      - 演示：

```
[root@localhost demo]# ls
hello.c  hello.i  hello.o  hello.s
[root@localhost demo]# gcc hello.o -o hello
[root@localhost demo]# ls
hello  hello.c  hello.i  hello.o  hello.s
[root@localhost demo]# ./hello 运行文件
helle~~ 输出结果
```

   6) 惯用：

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# gcc -c hello.c
[root@localhost demo]# ls
hello.c  hello.o
[root@localhost demo]# gcc hello.c -o hello
[root@localhost demo]# ls
hello  hello.c  hello.o
[root@localhost demo]# ./hello
helle~~
```

2. gcc 与 g++常用参数介绍：

1) -E

- -E 选项指示编译器仅对输入文件进行预处理。当这个选项被使用时，预处理器的输出被送到标准输出而不是储存在文件里。
- 演示：

```
[root@localhost demo]# gcc -E hello.c
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3
# 28 "/usr/include/stdio.h" 3
# 1 "/usr/include/features.h" 1 3
[illegible]
# 679 "/usr/include/stdio.h" 3

# 2 "hello.c" 2
int main()
{
  printf("helle~~\n");
}
```

2) -S

- -S 选项告诉 GCC 在为 C 代码产生了汇编语言文件后停止编译。GCC 产生的汇编语言文件的缺省扩展名是.s。
- 演示：

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# gcc -S hello.c
[root@localhost demo]# ls
hello.c  hello.s
```

3) -c

- -c 选项告诉 GCC 仅把源代码编译为目标代码。缺省时 GCC 建立的目标代码文件有一个.o 的扩展名。
- 演示：

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# gcc -c hello.c
[root@localhost demo]# ls
hello.c  hello.o
```

4) -o

- -o 选项来为将产生的可执行文件用指定的文件名。
- 演示：

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# gcc -c hello.c -o hi.o
[root@localhost demo]# ls
hello.c  hi.o
```

5) -O

- -O 选项告诉 GCC 对源代码进行基本优化，这些优化在大多数情况下都会使程序执行的更快，优化分为 4 个等级(-O0，-O1，-O2，-O3)。
  -O0 表示没有优化。
  -O1 为缺省值，主要进行跳转和延迟退栈两种优化。
  -O2 除了完成-O1 的优化之外，还进行一些额外的指令调整工作。
  -O3 除了完成-O2 的优化之外，还进行包括循环展开和其他一些与处理特性相关的优化工作。
- 演示：

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# gcc hello.c -o hello -O3
[root@localhost demo]# ls
hello  hello.c
```

其他优化等级的用法与演示中-O3 的用法一样。

6) -x

- 设定文件编译所使用的语言，使后缀名无效。
- 演示：

```
[root@localhost demo]# ls
demo.xxx
[root@localhost demo]# cat demo.xxx
#include<stdio.h>
int main()
{
  printf("Hello\n");
}
[root@localhost demo]# gcc -x c demo.xxx
[root@localhost demo]# ls
a.out  demo.xxx
[root@localhost demo]# ./a.out
Hello
```

使用C语言编译

7) -C

- 在预处理的时候，不删除注释信息。
- 演示：

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# cat hello.c
#include<stdio.h>
int main()
{
  printf("helle~~\n"); //shuchu helle
}
[root@localhost demo]# gcc -E hello.c -C
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3
[illegible]
# 2 "hello.c" 2
int main()
{
  printf("helle~~\n"); //shuchu helle
}
```

```
[root@localhost demo]# ls
hello.c
[root@localhost demo]# cat hello.c
#include<stdio.h>
int main()
{
  printf("helle~~\n"); //shuchu helle
}
[root@localhost demo]# gcc -E hello.c
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3
[illegible]
# 2 "hello.c" 2
int main()
{
  printf("helle~~\n");
}
```

8) -M

- 生成文件关联信息。包含目标文件所依赖的所有源代码。
- 演示：

```
[root@localhost demo]# gcc -M hello.c
hello.o: hello.c /usr/include/stdio.h /usr/include/features.h \
  /usr/include/sys/cdefs.h /usr/include/gnu/stubs.h \
  /usr/lib/gcc-lib/i386-redhat-linux/3.2.2/include/stddef.h \
  /usr/include/bits/types.h /usr/include/bits/wordsize.h \
  /usr/include/bits/typesizes.h /usr/include/libio.h \
  /usr/include/_G_config.h /usr/include/wchar.h /usr/include/bits/wchar.h \
  /usr/include/gconv.h \
  /usr/lib/gcc-lib/i386-redhat-linux/3.2.2/include/stdarg.h \
  /usr/include/bits/stdio_lim.h /usr/include/bits/sys_errlist.h
```

2. 编写 makefile：

1) 编写一个程序：

- 共 5 个文件，3 个.cpp 文件，2 个.h 文件。

```
[root@localhost test]# ls
main.cpp  printf1.cpp  printf1.h  printf2.cpp  printf2.h
```

- main.cpp：

```
[root@localhost test]# vim main.cpp
#include<iostream.h>
#include"printf1.h"
#include"printf2.h"
int main()
{
  printf1();
  printf2();
}
```

- printf1.cpp：

```
[root@localhost test]# vim printf1.cpp
#include<iostream.h>
#include"printf1.h"
void printf1()
{
  cout<<"Printf1"<<endl;
}
```

- printf1.h：

```
[root@localhost test]# vim printf1.h
void printf1();
```

- printf2.cpp：

```
[root@localhost test]# vim printf2.cpp
#include<iostream.h>
#include"printf2.h"
void printf2()
{
  cout<<"Printf2"<<endl;
}
```

- printf2.h：

```
[root@localhost test]# vim printf2.h
void printf2();
```

- 常规编译：
  - 汇编 main.cpp：

    ```
    [root@localhost test]# g++ -c main.cpp
    ```

  - 汇编 printf1.cpp

    ```
    [root@localhost test]# g++ -c printf1.cpp
    ```

  - 汇编 printf2.cpp

    ```
    [root@localhost test]# g++ -c printf2.cpp
    ```

  - 将 3 个 OBJ 文件链接一个链接文件上：

    ```
    [root@localhost test]# g++ main.o printf1.o printf2.o -o main
    ```

  - 运行：

    ```
    [root@localhost test]# ./main
    Printf1
    Printf2
    ```

  - 总共生成的文件：

    ```
    [root@localhost test]# ls
    main      main.o       printf1.h  printf2.cpp  printf2.o
    main.cpp  printf1.cpp  printf1.o  printf2.h
    ```

2) 运用 makefile 文件：

- 优点：

  由上例可知，照这样的编译方法，如果是一个项目的话，可能存在上百个文件，
  岂不是太麻烦了，所以要把编译过程写进一个文件中:makefile。

- 编写规则：
  - 以#号开始的为注释

- 文件依赖关系：
  对象：依赖项
  编译方式
- makefile 文本向右缩进时使用 TAB 键，不能用空格代替。

➢ 编写(以上面得程序为例)：

- 创建 makefile 文件：

```
[root@localhost test]# vim makefile
```

- 编写 makefile 文件：
  - 常用写法：

```
cc=g++                                  #指定编译器
exe=main                                #目标文件名
obj=main.o printf1.o printf2.o          #生成目标所使用的.o文件
$(exe):$(obj)                           #目标与.o文件的依赖关系
        $(cc) -o $(exe) $(obj)          #生成可执行文件
main.o:main.cpp printf1.h printf2.h     #main.o生成依赖main.cpp printf1.h printf2.h
        $(cc) -c main.cpp               #生成main.o
printf1.o:printf1.cpp printf1.h         #printf1.o生成依赖printf1.cpp printf1.h
        $(cc) -c printf1.cpp            #生成printf1.o
printf2.o:printf2.cpp printf2.h         #printf2.o生成依赖printf2.cpp printf2.h
        $(cc) -c printf2.cpp            #生成printf2.o
clean:                                  #clean伪目标
        rm-fr *.o $(exe)                #删除所有.o文件与目标文件
```

  - 也可以写成（文件一旦过多，修改也不容易）：

```
main:main.o printf1.o printf2.o
        g++ -o main main.o printf1.o printf2.o
main.o:main.cpp printf1.h printf2.h
        g++ -c main.cpp
printf1.o:printf1.cpp printf1.h
        g++ -c printf1.cpp
printf2.o:printf2.cpp printf2.h
        g++ -c printf2.cpp
clean:
        rm-fr *.o main
```

- 运行 makefile 文件：

```
[root@localhost test]# make
make: *** Warning: File `makefile' has modification time in the future (2010-04-26 21:06:5
9 > 2010-04-21 00:56:24.388489)
g++                                -c main.cpp            #生成main.o
In file included from /usr/include/c++/3.2.2/backward/iostream.h:31,
                 from main.cpp:1:
/usr/include/c++/3.2.2/backward/backward_warning.h:32:2: warning: #warning This file inclu
des at least one deprecated or antiquated header. Please consider using one of the 32 head
ers found in section 17.4.1.2 of the C++ standard. Examples include substituting the <X> h
eader for the <X.h> header for C++ includes, or <sstream> instead of the deprecated header
 <strstream.h>. To disable this warning use -Wno-deprecated.
g++                                -c printf1.cpp         #生成printf1.o
In file included from /usr/include/c++/3.2.2/backward/iostream.h:31,
                 from printf1.cpp:1:
/usr/include/c++/3.2.2/backward/backward_warning.h:32:2: warning: #warning This file inclu
des at least one deprecated or antiquated header. Please consider using one of the 32 head
ers found in section 17.4.1.2 of the C++ standard. Examples include substituting the <X> h
eader for the <X.h> header for C++ includes, or <sstream> instead of the deprecated header
 <strstream.h>. To disable this warning use -Wno-deprecated.
g++                                -o main                main.o pr
intf1.o printf2.o                       #生成可执行文件
make: warning:  Clock skew detected.  Your build may be incomplete.
```

- 生成 makefile 文件：

```
[root@localhost test]# ls
main      main.o    printf1.cpp  printf1.o    printf2.h
main.cpp  makefile  printf1.h    printf2.cpp  printf2.o
```

- 运行目标文件：

```
[root@localhost test]# ./main
Printf1
Printf2
```

- 删除生成文件：

```
[root@localhost test]# make clean
make: *** Warning: File `makefile' has modification time in the future (2010-04-26 21:06:5
9 > 2010-04-21 01:34:11.103424)
rm -fr *.o main                                    #删除所有.o文件与目标文件
make: warning:  Clock skew detected.  Your build may be incomplete.
[root@localhost test]# ls
main.cpp  makefile  printf1.cpp  printf1.h  printf2.cpp  printf2.h
```

周望博

2010 年 4 月 26 日星期一