gcc相关

基本用法：gcc [options] [filename]

不同参数的先后顺序对结果没有影响，但使用同类参数时需要考虑先后顺序

-o 输出文件名。默认输出可执行文件a.out

-g 产生符号调试工具(GNU的gdb)，对源代码调试时使用

-E 预编译，生成后缀为.i的预编译文件

-c 编译，不链接成可执行文件，由.c文件生成.o文件

-S 汇编，生成后缀为.s的汇编文件

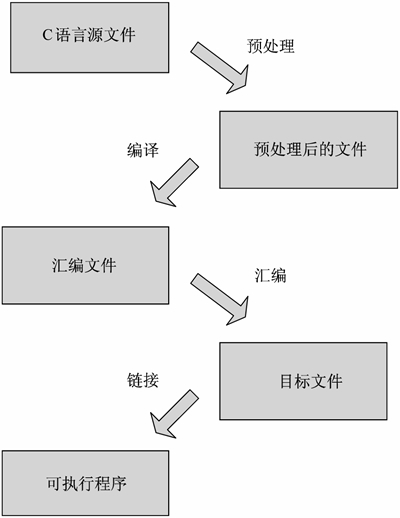
-I 指定include包含文件的目录

-L 指定库文件目录

-l 链接的库文件，有线链接动态库

-static 链接静态库

-Wall 警告信息



例子：

设源文件为test.c

#无选项，直接生成可执行文件a.out

gcc test.c

#-o 可执行文件test

gcc test.c -o test

#-E 生成预处理文件test.i

gcc -E test.c -o test.i

#-S 生成汇编文件test.s

gcc -S test.i -o test.s

#-c 生成目标文件test.o

gcc -c test.s -o test.o

#链接生成可执行文件test

gcc test.o -o test

#库文件链接，先编译成目标文件

gcc -c -I /usr/dev/mysql/include test.c -o test.o

#将目标文件链接成可执行文件，库默认前缀为lib，

#静态库后缀.a，动态库后缀.so，用-l链接库

gcc -L /usr/dev/mysql/lib -lmysqlclient test.o -o test

#链接静态库

gcc -L /usr/dev/mysql/lib -static -lmysqlclient test.o -o test

Makefile核心规则：

target ... : prerequisites ...  
            command

其中target是一个目标文件，可以是object file，也可以是可执行文件，还可以是一个标签。

prerequisites是生成target所需要的文件或目标

command是make需要执行的命令（任意shell命令）

这时一个文件依赖关系，target依赖于prerequisites中的文件，安装commong生成。如果prers中有一个以上文件比target新的化，command就会被执行。

example：

edit : main.o kbd.o command.o display.o /

           insert.o search.o files.o utils.o

            cc -o edit main.o kbd.o command.o display.o /

                       insert.o search.o files.o utils.o

main.o : main.c defs.h

            cc -c main.c

kbd.o : kbd.c defs.h command.h

            cc -c kbd.c

command.o : command.c defs.h command.h

            cc -c command.c

display.o : display.c defs.h buffer.h

            cc -c display.c

insert.o : insert.c defs.h buffer.h

            cc -c insert.c

search.o : search.c defs.h buffer.h

            cc -c search.c

files.o : files.c defs.h buffer.h command.h

            cc -c files.c

utils.o : utils.c defs.h

            cc -c utils.c

clean :

            rm edit main.o kbd.o command.o display.o /

               insert.o search.o files.o utils.o

说明：1.反斜杠”/”是换行符。2.命令以Tab键作为开头。

**变量**

可以定义变量，如

objects = main.o kdb.o command.o

以$(objects)的方式使用，如

edit: $(objects)

cc -o edit $(objects)

...

clean:

rm edit $(objects)

**自动推导**

只要make看到一个[.o]文件，它就会自动的把[.c]文件加在依赖关系中，如果make找到一个whatever.o，那么whatever.c，就会是whatever.o的依赖文件。并且 cc -c whatever.c 也会被推导出来，于是，我们的makefile再也不用写得这么复杂。我们的是新的makefile又出炉了：

    objects = main.o kbd.o command.o display.o /

              insert.o search.o files.o utils.o

    edit : $(objects)

            cc -o edit $(objects)

    main.o : defs.h

    kbd.o : defs.h command.h

    command.o : defs.h command.h

    display.o : defs.h buffer.h

    insert.o : defs.h buffer.h

    search.o : defs.h buffer.h

    files.o : defs.h buffer.h command.h

    utils.o : defs.h

    .PHONY : clean

    clean :

            rm edit $(objects)

这种方法，也就是make的“隐晦规则”。上面文件内容中，“.PHONY”表示，clean是个伪目标文件。

即然我们的make可以自动推导命令，那么我看到那堆[.o]和[.h]的依赖就有点不爽，那么多的重复的[.h]，能不能把其收拢起来，好吧，没有问题，这个对于make来说很容易，谁叫它提供了自动推导命令和文件的功能呢？来看看最新风格的makefile吧。

    objects = main.o kbd.o command.o display.o /

              insert.o search.o files.o utils.o

    edit : $(objects)

            cc -o edit $(objects)

    $(objects) : defs.h

    kbd.o command.o files.o : command.h

    display.o insert.o search.o files.o : buffer.h

    .PHONY : clean

    clean :

            rm edit $(objects)

这种风格，让我们的makefile变得很简单，但我们的文件依赖关系就显得有点凌乱了。

**清空规则**

更稳健的做法：

.PHONY : clean  
        clean :  
                -rm edit $(objects)