Linux 下 **make** 命令是系统管理员和程序员用的最频繁的命令之一。管理员用它通过命令行来编译和安装很多开源的工具，程序员用它来管理他们大型复杂的项目编译问题。本文我们将用一些实例来讨论 make 命令背后的工作机制。

**Make 如何工作的**

对于不知道背后机理的人来说，make 命令像命令行参数一样接收目标。这些目标通常存放在以 “Makefile” 来命名的特殊文件中，同时文件也包含与目标相对应的操作。更多信息，阅读关于 Makefiles 如何工作的系列文章。

当 make 命令第一次执行时，它扫描 Makefile 找到目标以及其依赖。如果这些依赖自身也是目标，继续为这些依赖扫描 Makefile 建立其依赖关系，然后编译它们。一旦主依赖编译之后，然后就编译主目标（这是通过 make 命令传入的）。

现在，假设你对某个源文件进行了修改，你再次执行 make 命令，它将只编译与该源文件相关的目标文件，因此，编译完最终的可执行文件节省了大量的时间。

**Make 命令实例**

下面是本文所使用的测试环境：

OS —— Ubunut 13.04

Shell —— Bash 4.2.45

Application —— GNU Make 3.81

下面是工程的内容：

$ ls

anotherTest.c Makefile test.c test.h

下面是 Makefile 的内容：

**all:** test

**test:** test.o anotherTest.o

gcc -Wall test.o anotherTest.o -o test

**test.o:** test.c

gcc -c -Wall test.c

**anotherTest.o:** anotherTest.c

gcc -c -Wall anotherTest.c

**clean:**

rm -rf \*.o test

现在我们来看 Linux 下一些 make 命令应用的实例：

**1. 一个简单的例子**

为了编译整个工程，你可以简单的使用 make 或者在 make 命令后带上目标 all。

$ make

gcc -c -Wall test.c

gcc -c -Wall anotherTest.c

gcc -Wall test.o anotherTest.o -o test

你能看到 make 命令第一次创建的依赖以及实际的目标。

如果你再次查看目录内容，里面多了一些 .o 文件和执行文件：

$ ls

anotherTest.c anotherTest.o Makefile test test.c test.h test.o

现在，假设你对 test.c 文件做了一些修改，重新使用 make 编译工程：

$ make

gcc -c -Wall test.c

gcc -Wall test.o anotherTest.o -o test

你可以看到只有 test.o 重新编译了，然而另一个 Test.o 没有重新编译。

现在清理所有的目标文件和可执行文件 test，你可以使用目标 clean:

$ make clean

rm -rf \*.o test

$ ls

anotherTest.c Makefile test.c test.h

你可以看到所有的 .o 文件和执行文件 test 都被删除了。

**2. 通过 -B 选项让所有目标总是重新建立**

到目前为止，你可能注意到 make 命令不会编译那些自从上次编译之后就没有更改的文件，但是，如果你想覆盖 make 这种默认的行为，你可以使用 -B 选项。

下面是个例子：

$ make

make: Nothing to be **done for** `all’.

$ make -B

gcc -c -Wall test.c

gcc -c -Wall anotherTest.c

gcc -Wall test.o anotherTest.o -o test

你可以看到尽管 make 命令不会编译任何文件，然而 make -B 会强制编译所有的目标文件以及最终的执行文件。

**3. 使用 -d 选项打印调试信息**

如果你想知道 make 执行时实际做了什么，使用 -d 选项。

这是一个例子：

$ make -d | more

GNU Make 3.81

Copyright **(**C**)** 2006 Free Software Foundation, Inc.

This is free software; see the source **for** copying conditions.

There is NO warranty; not even **for** MERCHANTABILITY or FITNESS FOR A

PARTICULAR PURPOSE.

This program built **for** x86\_64-pc-linux-gnu

Reading makefiles…

Reading makefile `Makefile’…

Updating makefiles….

Considering target file `Makefile’.

Looking **for** an implicit rule **for** `Makefile’.

Trying pattern rule with stem `Makefile’.

Trying implicit prerequisite `Makefile.o’.

Trying pattern rule with stem `Makefile’.

Trying implicit prerequisite `Makefile.c’.

Trying pattern rule with stem `Makefile’.

Trying implicit prerequisite `Makefile.cc’.

Trying pattern rule with stem `Makefile’.

Trying implicit prerequisite `Makefile.C’.

Trying pattern rule with stem `Makefile’.

Trying implicit prerequisite `Makefile.cpp’.

Trying pattern rule with stem `Makefile’.

--More--

这是很长的输出，你也看到我使用了 more 命令来一页一页显示输出。

**4. 使用 -C 选项改变目录**

你可以为 make 命令提供不同的目录路径，在寻找 Makefile 之前会切换目录的。

这是一个目录，假设你就在当前目录下:

$ ls

file file2 frnd frnd1.cpp log1.txt log3.txt log5.txt

file1 file name with spaces frnd1 frnd.cpp log2.txt log4.txt

但是你想运行的 make 命令的 Makefile 文件保存在 ../make-dir/ 目录下，你可以这样做：

$ make -C ../make-dir/

make: Entering directory `/home/himanshu/practice/make-dir’

make: Nothing to be **done for** `all’.

make: Leaving directory `/home/himanshu/practice/make-dir

你能看到 make 命令首先切到特定的目录下，在那执行，然后再切换回来。

**5. 通过 -f 选项将其它文件看作 Makefile**

如果你想将重命名 Makefile 文件，比如取名为 my\_makefile 或者其它的名字，我们想让 make 将它也当成 Makefile，可以使用 -f 选项。

make -f my\_makefile

通过这种方法，make 命令会选择扫描 my\_makefile 来代替 Makefile。