如何寫 Makefile 文件

Makefile的規則。

```
target ... : prerequisites ...
command
...
```

target也就是一個目標文件，可以是Object File，也可以是執行文件。還可以是一個標籤（Label），對於標籤這種特性，在後續的“偽目標”章節中會有敘述。

prerequisites就是，要生成那個target所需要的文件或是目標。

command也就是make需要執行的命令。（任意的Shell命令）

這 是一個文件的依賴關係，也就是說，target這一個或多個的目標文件依賴於prerequisites中的文件，其生成規則定義在command中。說 白一點就是說，prerequisites中如果有一個以上的文件比target文件要新的話，command所定義的命令就會被執行。這就是 Makefile的規則。也就是Makefile中最核心的內容。

說到底，Makefile的東西就是這樣一點，好像我的這篇文檔也該結束了。呵呵。還不盡然，這是Makefile的主線和核心，但要寫好一個Makefile還不夠，我會以後面一點一點地結合我的工作經驗給你慢慢到來。內容還多著呢。：）

二、一個示例
正如前面所說的，如果一個工程有3個頭文件，和8個C文件，我們為了完成前面所述的那三個規則，我們的Makefile應該是下面的這個樣子的。

```
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
```

反 斜槓（\）是換行符的意思。這樣比較便於Makefile的易讀。我們可以把這個內容保存在文件為“Makefile”或“makefile”的文件中， 然後在該目錄下直接輸入命令“make”就可以生成執行文件edit。如果要刪除執行文件和所有的中間目標文件，那麼，只要簡單地執行一下 “make clean”就可以

了。

在這個makefile中，目標文件（target）包含：執行文件edit和中間目標文 件（*.o），依賴文件（prerequisites）就是冒號後面的那些 .c 文件和 .h文件。每一個 .o 文件都有一組依賴文件，而這些 .o 文件又是執行文件 edit 的依賴文件。 依賴關係的實質上就是說明了目標文件是由哪些文件生成的，換言之，目標文件是哪些文件更新的。

在定義好依賴關係後，後續的那一行定義了如 何生成目標文件的操作系統命令，一定要以一個Tab鍵作為開頭。記住，make並不管命令是怎麼工作的，他只管執行所定義的命令。make會比較 targets文件和prerequisites文件的修改日期，如果prerequisites文件的日期要比targets文件的日期要新，或者target不存在的話，那麼，make就會執行後續定義的命令。

這裡要說明一點的是，clean不是一個文件，它只不過是一個動作名 字，有點像C語言中的lable一樣，其冒號後什麼也沒有，那麼，make就不會自動去找文件的依賴性，也就不會自動執行其後所定義的命令。要執行其後的 命令，就要在make命令後明顯得指出這個lable的名字。這樣的方法非常有用，我們可以在一個makefile中定義不用的編譯或是和編譯無關的命 令，比如程序的打包，程序的備份，等等。

三、make是如何工作的
在默認的方式下，也就是我們只輸入make命令。那麼，

1、make會在當前目錄下找名字叫“Makefile”或“makefile”的文件。

2、如果找到，它會找文件中的第一個目標文件（target），在上面的例子中，他會找到“edit”這個文件，並把這個文件作為最終的目標文件。

3、如果edit文件不存在，或是edit所依賴的後面的.o文件的文件修改時間要比edit這個文件新，那麼，他就會執行後面所定義的命令來生成edit這個文件。

4、如果edit所依賴的.o文件也存在，那麼make會在當前文件中找目標為.o文件的依賴性，如果找到則再根據那一個規則生成.o文件。（這有點像一個堆棧的過程）

5、當然，你的C文件和H文件是存在的啦，於是make會生成.o文件，然後再用.o文件生命make的終極任務，也就是執行文件edit了。

這 就是整個make的依賴性，make會一層又一層地去找文件的依賴關係，直到最終編譯出第一個目標文件。在找尋的過程中，如果出現錯誤，比如最後被依賴的文件找不到，那麼make就會直接退出，並報錯，而對於所定義的命令的錯誤，或是編譯不成功，make根本不理。make只管文件的依賴性，即，如果在我 找了依賴關係之後，冒號後面的文件還是不在，那麼對不起，我就不工作啦。

通過上述分析，我們知道，像clean這種，沒有被第一個目標文件直接或間接關聯，那麼它後面所定義的命令將不會被自動執行，不過，我們可以顯示要make執行。即命令——“make clean”，以此來清除所有的目標文件，以便重編譯。

於 是在我們編程中，如果這個工程已被編譯過了，當我們修改了其中一個源文件，比如file.c，那麼根據我們的依賴性，我們的目標file.o會被重編譯 （也就是在這個依性關係後面所定義的命令），於是file.o的文件也是最新的啦，於是file.o的文件修改時間要比edit要新，所以edit也會被 重新鏈接了（詳見edit目標文件後定義的命令）。

而如果我們改變了“command.h”，那麼，kdb.o、command.o和files.o都會被重編譯，並且，edit會被重鏈接。

四、makefile中使用變量
在上面的例子中，先讓我們看看edit的規則：

```
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
```

我 們可以看到[.o]文件的字符串被重複了兩次，如果我們的工程需要加入一個新的[.o]文件，那麼我們需要在兩個地方加（應該是三個地方，還有一個地方在 clean中）。當然，我們的makefile並不複雜，所以在兩個地方加也不累，但如果makefile變得複雜，那麼我們就有可能會忘掉一個需要加入 的地方，而導致編譯失敗。所以，為了makefile的易維護，在makefile中我們可以使用變量。makefile的變量也就是一個字符串，理解成 C語言中的宏可能會更好。

比如，我們聲明一個變量，叫objects, OBJECTS, objs, OBJS, obj, 或是 OBJ，反正不管什麼啦，只要能夠表示obj文件就行了。我們在makefile一開始就這樣定義：

```
objects = main.o kbd.o command.o display.o
insert.o search.o files.o utils.o
```

於是，我們就可以很方便地在我們的makefile中以“$(objects)”的方式來使用這個變量了，於是我們的改良版makefile就變成下面這個樣子：

```
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 : 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 $(objects)
```

於是如果有新的.o 文件加入，我們只需簡單地修改一下 objects 變量就可以了。

關於變量更多的話題，我會在後續給你一一道來。

五、讓make自動推導

GNU的make很強大，它可以自動推導文件以及文件依賴關係後面的命令，於是我們就沒必要去在每一個[.o]文件後都寫上類似的命令，因為，我們的make會自動識別，並自己推導命令。

只 要make看到一個[.o]文件，它就會自動的把[.c]文件加在依賴關係中，如果make找到一個whatever.o，那麼whatever.c，就 會是whatever.o的依賴文件。並且 cc -c whatever.c 也會被推導出來，於是，我們的 makefile再也不用寫得這麼複雜。我們的是新的makefile又出爐了。

```
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)
```

這種方法，也就是make的“隱晦規則”。上面文件內容中，“.PHONY”表示，clean是個偽目標文件。

關於更為詳細的“隱晦規則”和“偽目標文件”，我會在後續給你一一道來。

六、另類風格的makefile
即然我們的make可以自動推導命令，那麼我看到那堆[.o]和[.h]的依賴就有點不爽，那麼多的重複的[.h]，能不能把其收攏起來，好吧，沒有問題，這個對於make來說很容易，誰叫它提供了自動推導命令和文件的功能呢？來看看最新風格的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)
```

這種風格，讓我們的makefile變得很簡單，但我們的文件依賴關係就顯得有點凌亂了。魚和熊掌不可兼得。還看你的喜好了。我是不喜歡這種風格的，一是文件的依賴關係看不清楚，二是如果文件一多，要加入幾個新的.o文件，那就理不清楚了。

七、清空目標文件的規則
每個Makefile中都應該寫一個清空目標文件（.o和執行文件）的規則，這不僅便於重編譯，也很利於保持文件的清潔。這是一個“修養”（呵呵，還記得我的《編程修養》嗎）。一般的風格都是：

```
clean:
```

```
rm edit $(objects)
```

更為穩健的做法是：

```
.PHONY : clean

clean :

-rm edit $(objects)
```

前面說過，.PHONY意思表示clean是一個“偽目標”，。而在rm命令前面加了一個小減號的意思就是，也許某些文件出現問題，但不要管，繼續做後面的事。當然，clean的規則不要放在文件的開頭，不然，這就會變成make的默認目標，相信誰也不願意這樣。不成文的規矩是——“clean從來都是放在 文件的最後”。