跟我一起寫**Makefile:make**運行

出自**Ubuntu**中文

*導 |概述 + MakeFile介紹 + 書寫規則 + 書寫命令 + 使用變量 + 使用條件判斷 |
*航 |使用函數 + **make**運行 + 隱含規則 + 使用make更新函數庫文件 + 後序 |
*

make 的運行

一般來說，最簡單的就是直接在命令行下輸入make命令，
make命令會找當前目錄的makefile來執行，一切都是自動的。
但也有時你也許只想讓make重編譯某些文件，而不是整個工程，而又有的時候你有幾套編譯

規則，你想在不同的時候使用不同的編譯規則，等等。本章節就是講述如何使用 make命令的。

make的退出碼

make命令執行後有三個退出碼：

```
0 —— 表示成功执行。
1 —— 如果make运行时出现任何错误，其返回1。
2 —— 如果你使用了make的“-q”选项，并且make使得一些目标不需要更新，那么返回2。
```

Make的相關參數我們會在後續章節中講述。

指定Makefile

前面我們說過，GNU make找尋默認的Makefile的規則是在當前目錄下依次找三個文件——“GNUmakefile”、“makefile”和“Makefile”。其按順序找這三個文件，一旦找到，就開始讀取這個文件並執行。

當前，我們也可以給make命令指定一個特殊名字的Makefile。要達到這個功能，我們要使用make的“-f”或是“--file”參數（“--makefile”參數也行）。例如，我們有個makefile的名字是“hchen.mk”，那麼，我們可以這樣來讓make來執行這個文件：

```
make -f hchen.mk
```

如果在make的命令行是，你不只一次地使用了“-f”參數，那麼，所有指定的makefile將會被連在一起傳遞給make執行。

指定目標

一般來說，make的最終目標是makefile中的第一個目標，而其它目標一般是由這個目標連帶出來的。這是make的默認行為。當然，一般來說，你的makefile中的第一個目標是由許多個目標組成，你可以指示make，讓其完成你所指定的目標。要達到這一目的很簡單，需在make命令后直接跟目標的名字就可以完成（如前面提到的“make clean”形式）

任何在makefile中的目標都可以被指定成終極目標，但是除了以“-”打頭，或是包含了“=”的目標，因為有這些字符的目標，會被解析成命令行參數或是變量。甚至沒有被我們明確寫出來的目標也可以成為make的終極目標，也就是說，只要make可以找到其隱含規則推導規則，那麼這個隱含目標同樣可以被指定成終極目標。

有一個make的環境變量叫“MAKECMDGOALS”，這個變量中會存放你所指定的終極目標的列表，如果在命令行上，你沒有指定目標，那麼，這個變量是空值。這個變量可以讓你使用在一些比較特殊的情形下。比如下面的例子：

```
sources = foo.c bar.c
ifneq ( $(MAKECMDGOALS),clean)
include $(sources:.c=.d)
endif
```

基於上面的這個例子，只要我們輸入的命令不是“make clean”，那麼makefile會自動包含“foo.d”和“bar.d”這兩個makefile。

使用指定終極目標的方法可以很方便地讓我們編譯我們的程序，例如下面這個例子：

```
.PHONY: all
all: prog1 prog2 prog3 prog4
```

從這個例子中，我們可以看到，這個makefile中有四個需要編譯的程序——“prog1”，“prog2”，“prog3”和“prog4”，我們可以使用“make all”命令來編譯所有的目標（如果把all置成第一個目標，那麼只需執行“make”），我們也可以使用“make prog2”來單獨編譯目標“prog2”。

即然make可以指定所有makefile中的目標，那麼也包括“偽目標”，於是我們可以根據這種性質來讓我們的makefile根據指定的不同的目標來完成不同的事。在Unix世界中，軟件發布時，特別是GNU這種開源軟件的發布時，其makefile都包含了編譯、安裝、打包等功能。我們可以參照這種規則來書寫我們的makefile中的目標。

“all”
　　這個偽目標是所有目標的目標，其功能一般是編譯所有的目標。
clean”
　　這個偽目標功能是刪除所有被make創建的文件。
“install”
　　這個偽目標功能是安裝已編譯好的程序，其實就是把目標執行文件拷貝到指定的目標中去。
print”
　　這個偽目標的功能是例出改變過的源文件。
“tar”
　　這個偽目標功能是把源程序打包備份。也就是一個tar文件。
“dist”
　　這個偽目標功能是創建一個壓縮文件，一般是把tar文件壓成Z文件。或是gz文件。
TAGS”
　　這個偽目標功能是更新所有的目標，以備完整地重編譯使用。
“check”和“test”
　　這兩個偽目標一般用來測試makefile的流程。

當然一個項目的makefile中也不一定要書寫這樣的目標，這些東西都是GNU的東西，但是我想，GNU搞出這些東西一定有其可取之處（等你的 UNIX下的程序文件一多時你就會發現這些功能很有用了），這裏只不過是說明了，如果你要書寫這種功能，最好使用這種名字命名你的目標，這樣規範一些，規範的好處就是——不用解釋，大家都明白。而且如果你的makefile中有這些功能，一是很實用，二是可以顯得你的makefile很專業（不是那種初學者的作品）。

檢查規則

有時候，我們不想讓我們的makefile中的規則執行起來，我們只想檢查一下我們的命令，或是執行的序列。於是我們可以使用make命令的下述參數：

“-n” “--just-print” “--dry-run” “--recon” 不執行參數，這些參數只是打印命令，不管目標是否更新，把規則和連帶規則下的命令打印出來，但不執行，這些參數對於我們調試makefile很有用處。

“-t” “--touch” 這個參數的意思就是把目標文件的時間更新，但不更改目標文件。也就是說，make假裝編譯目標，但不是真正的編譯目標，只是把目標變成已編譯過的狀態。

“-q” “--question” 這個參數的行為是找目標的意思，也就是說，如果目標存在，那麼其什麼也不會輸出，當然也不會執行編譯，如果目標不存在，其會打印出一條出錯信息。

“-W <file>;” “--what-if=<file>;” “--assume-new=<file>;” “--new-file=<file>;” 這個參數需要指定一個文件。一般是是源文件（或依賴文件），Make會根據規則推導來運行依賴於這個文件的命令，一般來說，可以和“-n”參數一同使用，來查看這個依賴文件所發生的規則命令。

另外一個很有意思的用法是結合“-p”和“-v”來輸出makefile被執行時的信息（這個將在後面講述）。

make的參数

下面列舉了所有GNU make 3.80版的參數定義。其它版本和產商的make大同小異，不過其它產商的make的具體參數還是請參考各自的產品文檔。

“-b” “-m” 這兩個參數的作用是忽略和其它版本make的兼容性。

“-B” “--always-make” 認為所有的目標都需要更新（重編譯）。

“-C <dir>” “--directory=<dir>” 指定讀取makefile的目錄。如果有多個“-C”參數，make的解釋是後面的路徑以前面的作為相對路徑，並以最後的目錄作為被指定目錄。如：“make －C ~hchen/test －C prog”等價于“make －C ~hchen/test/prog”。

“—debug[=<options>]” 輸出make的調試信息。它有幾種不同的級別可供選擇，如果沒有參數，那就是輸出最簡單的調試信息。下面是<options>的取值：

```
a —— 也就是all，输出所有的调试信息。（会非常的多）
b —— 也就是basic，只输出简单的调试信息。即输出不需要重编译的目标。
v —— 也就是verbose，在b选项的级别之上。输出的信息包括哪个makefile被解析，不需要被重编译的依赖文件（或是依赖目标）等。
i —— 也就是implicit，输出所以的隐含规则。
j —— 也就是jobs，输出执行规则中命令的详细信息，如命令的PID、返回码等。
m —— 也就是makefile，输出make读取makefile，更新makefile，执行makefile的信息。
```

“-d” 相當於“--debug=a”。

“-e” “--environment-overrides” 指明環境變量的值覆蓋makefile中定義的變量的值。

“-f=<file>” “--file=<file>” “--makefile=<file>” 指定需要執行的makefile。

“-h” “--help” 顯示幫助信息。

“-i” “--ignore-errors” 在執行時忽略所有的錯誤。

“-I <dir>” “--include-dir=<dir>” 指定一個被包含makefile的搜索目標。可以使用多個“-I”參數來指定多個目錄。

“-j [<jobsnum>]” “--jobs[=<jobsnum>]” 指同時運行命令的個數。如果沒有這個參數，make運行命令時能運行多少就運行多少。如果有一個以上的“-j”參數，那麼僅最後一個“-j”才是有效的。（注意這個參數在MS-DOS中是無用的）

“-k” “--keep-going” 出錯也不停止運行。如果生成一個目標失敗了，那麼依賴於其上的目標就不會被執行了。

“-l <load>” “--load-average[=<load]” “—max-load[=<load>]” 指定make運行命令的負載。

“-n” “--just-print” “--dry-run” “--recon” 僅輸出執行過程中的命令序列，但並不執行。

“-o <file>” “--old-file=<file>” “--assume-old=<file>” 不重新生成的指定的<file>，即使這個目標的依賴文件新於它。

“-p” “--print-data-base” 輸出makefile中的所有數據，包括所有的規則和變量。這個參數會讓一個簡單的makefile都會輸出一堆信息。如果你只是想輸出信息而不想執行makefile，你可以使用“make -qp”命令。如果你想查看執行makefile前的預設變量和規則，你可以使用“make －p －f /dev/null”。這個參數輸出的信息會包含着你的makefile文件的文件名和行號，所以，用這個參數來調試你的 makefile會是很有用的，特別是當你的環境變量很複雜的時候。

“-q” “--question” 不運行命令，也不輸出。僅僅是檢查所指定的目標是否需要更新。如果是0則說明要更新，如果是2則說明有錯誤發生。

“-r” “--no-builtin-rules” 禁止make使用任何隱含規則。

“-R” “--no-builtin-variabes” 禁止make使用任何作用於變量上的隱含規則。

“-s” “--silent” “--quiet” 在命令運行時不輸出命令的輸出。

“-S” “--no-keep-going” “--stop” 取消“-k”選項的作用。因為有些時候，make的選項是從環境變量“MAKEFLAGS”中繼承下來的。所以你可以在命令行中使用這個參數來讓環境變量中的“-k”選項失效。

“-t” “--touch” 相當於UNIX的touch命令，只是把目標的修改日期變成最新的，也就是阻止生成目標的命令運行。

“-v” “--version” 輸出make程序的版本、版權等關於make的信息。

“-w” “--print-directory” 輸出運行makefile之前和之後的信息。這個參數對於跟蹤嵌套式

調用make時很有用。

“--no-print-directory” 禁止“-w”選項。

“-W <file>” “--what-if=<file>” “--new-file=<file>” “--assume-file=<file>” 假定目標<file>;需要更新，如果和“-n”選項使用，那麼這個參數會輸出該目標更新時的運行動作。如果沒有“-n”那麼就像運行UNIX的“touch”命令一樣，使得<file>;的修改時間為當前時間。

“--warn-undefined-variables” 只要make發現有未定義的變量，那麼就輸出警告信息。