跟我一起寫Makefile:書寫命令

出自**Ubuntu**中文

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

目錄

書寫命令

每條規則中的命令和操作系統Shell的命令行是一致的。make會一按順序一條一條的執行命令，每條命令的開頭必須以[Tab]鍵開頭，除非，命令是緊跟在依賴規則後面的分號后的。在命令行之間中的空格或是空行會被忽略，但是如果該空格或空行是以Tab鍵開頭的，那麼make會認為其是一個空命令。

我們在UNIX下可能會使用不同的Shell，但是make的命令默認是被“/bin/sh”——UNIX的標準Shell解釋執行的。除非你特別指定一個其它的Shell。Makefile中，“#”是註釋符，很像C/C++中的“//”，其後的本行字符都被註釋。

顯示命令

通常，make會把其要執行的命令行在命令執行前輸出到屏幕上。當我們用“@”字符在命令行前，那麼，這個命令將不被make顯示出來，最具代表性的例子是，我們用這個功能來像屏幕顯示一些信息。如：

```
@echo 正在编译XXX模块.....
```

當make執行時，會輸出“正在編譯XXX模塊.....”字串，但不會輸出命令，如果沒有“@”，那麼，make將輸出：

```
echo 正在编译XXX模块.....
正在编译XXX模块.....
```

如果make執行時，帶入make參數“-n”或“--just-print”，那麼其只是顯示命令，但不會執行命令，這個功能很有利於我們調試我們的Makefile，看看我們書寫的命令是執行起來是什麼樣子的或是什麼順序的。

而make參數“-s”或“--slient”則是全面禁止命令的顯示。

命令執行

當依賴目標新于目標時，也就是當規則的目標需要被更新時，make會一條一條的執行其後的命令。需要注意的是，如果你要讓上一條命令的結果應用在下一條命令時，你應該使用分號分隔這兩條命令。比如你的第一條命令是cd命令，你希望第二條命令得在cd之後的基礎上運行，那麼你就不能把這兩條命令寫在兩行上，而應該把這兩條命令寫在一行上，用分號分隔。如：

示例一：

```
exec:
        cd /home/hchen
        pwd
```

示例二：

```
exec:
        cd /home/hchen; pwd
```

當我們執行“make exec”時，第一個例子中的cd沒有作用，pwd會打印出當前的Makefile目錄，而第二個例子中，cd就起作用了，pwd會打印出“/home/hchen”。

make一般是使用環境變量SHELL中所定義的系統Shell來執行命令，默認情況下使用UNIX的

標準Shell——/bin/sh來執行命令。但在MS-DOS下有點特殊，因為MS-DOS下沒有SHELL環境變量，當然你也可以指定。如果你指定了UNIX風格的目錄形式，首先，make會在SHELL所指定的路徑中找尋命令解釋器，如果找不到，其會在當前盤符中的當前目錄中尋找，如果再找不到，其會在PATH環境變量中所定義的所有路徑中尋找。MS-DOS中，如果你定義的命令解釋器沒有找到，其會給你的命令解釋器加上諸如“.exe”、“.com”、“.bat”、“.sh”等後綴。

命令出錯

每當命令運行完后，make會檢測每個命令的返回碼，如果命令返回成功，那麼make會執行下一條命令，當規則中所有的命令成功返回后，這個規則就算是成功完成了。如果一個規則中的某個命令出錯了（命令退出碼非零），那麼make就會終止執行當前規則，這將有可能終止所有規則的執行。

有些時候，命令的出錯並不表示就是錯誤的。例如mkdir命令，我們一定需要建立一個目錄，如果目錄不存在，那麼mkdir就成功執行，萬事大吉，如果目錄存在，那麼就出錯了。我們之所以使用mkdir的意思就是一定要有這樣的一個目錄，於是我們就不希望mkdir出錯而終止規則的運行。

為了做到這一點，忽略命令的出錯，我們可以在Makefile的命令行前加一個減號“-”（在Tab鍵之後），標記為不管命令出不出錯都認為是成功的。如：

```
clean:
        -rm -f *.o
```

還有一個全局的辦法是，給make加上“-i”或是“--ignore-errors”參數，那麼，Makefile中所有命令都會忽略錯誤。而如果一個規則是以“.IGNORE”作為目標的，那麼這個規則中的所有命令將會忽略錯誤。這些是不同級別的防止命令出錯的方法，你可以根據你的不同喜歡設置。

還有一個要提一下的make的參數的是“-k”或是“--keep-going”，這個參數的意思是，如果某規則中的命令出錯了，那麼就終目該規則的執行，但繼續執行其它規則。

嵌套執行**make**

在一些大的工程中，我們會把我們不同模塊或是不同功能的源文件放在不同的目錄中，我們可以在每個目錄中都書寫一個該目錄的Makefile，這有利於讓我們的Makefile變得更加地簡潔，而不至於把所有的東西全部寫在一個Makefile中，這樣會很難維護我們的Makefile，這個技術對於我們模塊編譯和分段編譯有着非常大的好處。

例如，我們有一個子目錄叫subdir，這個目錄下有個Makefile文件，來指明了這個目錄下文件的編譯規則。那麼我們總控的Makefile可以這樣書寫：

```
subsystem:
        cd subdir && $(MAKE)
```

其等價于：

```
subsystem:
	$(MAKE) -C subdir
```

定義$(MAKE)宏變量的意思是，也許我們的make需要一些參數，所以定義成一個變量比較利於維護。這兩個例子的意思都是先進入“subdir”目錄，然後執行make命令。

我們把這個Makefile叫做“總控Makefile”，總控Makefile的變量可以傳遞到下級的Makefile中（如果你顯示的聲明），但是不會覆蓋下層的Makefile中所定義的變量，除非指定了“-e”參數。

如果你要傳遞變量到下級Makefile中，那麼你可以使用這樣的聲明：

```
export <variable ...>;
```

如果你不想讓某些變量傳遞到下級Makefile中，那麼你可以這樣聲明：

```
unexport <variable ...>;
```

如：

示例一：

```
export variable = value
```

其等價于：

```
variable = value
export variable
```

其等價于：

```
export variable  := value
```

其等價于：

```
variable  := value
export variable
```

示例二：

```
export variable += value
```

其等價于：

```
variable += value
export variable
```

如果你要傳遞所有的變量，那麼，只要一個export就行了。後面什麼也不用跟，表示傳遞所有的變量。

需要注意的是，有兩個變量，一個是SHELL，一個是MAKEFLAGS，這兩個變量不管你是否export，其總是要傳遞到下層 Makefile中，特別是MAKEFILES變量，其中包含了make的參數信息，如果我們執行“總控Makefile”時有make參數或是在上層 Makefile中定義了這個變量，那麼MAKEFILES變量將會是這些參數，並會傳遞到下層Makefile中，這是一個系統級的環境變量。

但是make命令中的有幾個參數並不往下傳遞，它們是“-C”,“-f”,“-h” “-o”和“-W”（有關Makefile參數的細節將在後面說明），如果你不想往下層傳遞參數，那麼，你可以這樣來：

```
subsystem:
	cd subdir && $(MAKE) MAKEFLAGS=
```

如果你定義了環境變量MAKEFLAGS，那麼你得確信其中的選項是大家都會用到的，如果其中有“-t”,“-n”,和“-q”參數，那麼將會有讓你意想不到的結果，或許會讓你異常地恐慌。

還有一個在“嵌套執行”中比較有用的參數，“-w”或是“--print-directory”會在make的過程中輸出一些信息，讓你看到目前的工作目錄。比如，如果我們的下級make目錄是“/home/hchen/gnu/make”，如果我們使用“make -w”來執行，那麼當進入該目錄時，我們會看到：

```
make: Entering directory `/home/hchen/gnu/make'.
```

而在完成下層make后離開目錄時，我們會看到：

```
make: Leaving directory `/home/hchen/gnu/make'
```

當你使用“-C”參數來指定make下層Makefile時，“-w”會被自動打開的。如果參數中有“-s”（“--slient”）或是“--no-print-directory”，那麼，“-w”總是失效的。

定義命令包

如果Makefile中出現一些相同命令序列，那麼我們可以為這些相同的命令序列定義一個變量。定義這種命令序列的語法以“define”開始，以“endef”結束，如：

```
define run-yacc
yacc $(firstword $^)
mv y.tab.c $@
endef
```

這裏，“run-yacc”是這個命令包的名字，其不要和Makefile中的變量重名。在“define”和“endef”中的兩行就是命令序列。這個命令包中的第一個命令是運行Yacc程序，因為Yacc程序總是生成“y.tab.c”的文件，所以第二行的命令就是把這個文件改改名字。還是把這個命令包放到一個示例中來看看吧。

```
foo.c  : foo.y
       $(run-yacc)
```

我們可以看見，要使用這個命令包，我們就好像使用變量一樣。在這個命令包的使用中，命令包“run-yacc”中的“$^”就是“foo.y”， “$@”就是“foo.c”（有關這種以“$”開頭的特殊變量，我們會在後面介紹），make在執行命令包時，命令包中的每個命令會被依次獨立執行。