

shell 命令稿的 撰寫

在前面的章節中，我們看過了 *bash* 的各個組成要件，也瞭解到了使用它們來撰寫 shell 命令稿的方法。如果你曾經用過其他程式語言，那麼你將會瞭解到「只是撰寫一段程式碼來完成工作」與「撰寫一段程式碼來做工作並考慮到它的可維護性以及是否符合我們所謂的“好典範”(good practice)」是有所不同的。

本章將會對「好典範以及具維護性之 shell 命令稿的撰寫」做出簡單的說明，並且提供若干有用的提示和訣竅，讓你得以用來簡化 shell 命令稿的撰寫工作。

11.1 那是在做什麼啊？

六個月之前，你撰寫了一支 100 列長的 shell 命令稿。當時你對它瞭若指掌，但是而今看到它，你心理可能會納悶「那是在做什麼啊？」，這是程式設計人員的通病 — 尤其是撰寫 shell 命令稿的程式設計人員。不幸的是，shell 被發展成過份使用晦澀的標點符號。這麼做雖然可以让你少打些字，但是對可讀性一點幫助也沒有。重點應該是如何讓你的程式碼更具可讀性。

11.1.1 註解

shell 命令稿撰寫的第一項守則是「為程式碼加上註解」。你應該從命令稿的開頭就這麼做，即使你的命令稿沒有幾列程式碼。當有更多的功能被加進來時，shell 命令稿一般會從數列程式碼成長到數百列程式碼，所以我們最好養成一開始就為程式碼加上註解的好習慣。

首先，為你的命令稿加上主標頭（**header**）或標題（**banner**）。標頭中的資訊（至少）應該提到命令稿的功能。下面是一個命令稿標頭實例：

```
#!/bin/bash
#####
# 名稱:graphconv.sh
#
# 將圖形檔從一個格式轉換成另一個。
#
# 用法:graphconv.sh <input-file> <output-file>
#
# 作者:C. Newham
# 日期:2004/12/02
#####
```

這個主標頭提供了命令稿的名稱、功能摘要、用法資訊、作者以及日期。

在使用原碼控制系統（例如 **CVS**）的狀況下，如果命令稿在交付時會儲存作者和日期，那麼你可以不必提供這些資訊。在不使用此類系統的狀況下，我們強烈建議你不僅要提供以上資訊，而且還要在標頭中加入額外的資料，像是進行修改的日期和作者。

不論你使用的是何種系統，務必維持標題格式的一致性。

每個函式也應該具有一個標頭。如果它是一個獨立的函式，則應該具備如上所示的主標頭。如果它是命令稿內部所使用的函式，則應該具備一個較簡單的標題，用來描述它的功能、參數以及傳回值，例如：

```
# 變更副檔名
#
# 參數:$infile -- 原來的檔案名稱
#
# 傳回值:具新副檔名的檔案名稱
#
function change_filename()
...
```

為了說明程式碼在做什麼，你還應該在程式碼中頻繁地使用註解。儘管不一定要這麼做，不過你最好是依照程式碼的流程來擺放註解，而不要將它們全都擺在同一列上，而且最好為變數的宣告加註解，而不要為變數加註解：

```
startup_dir=/home/startup/ # 包含啟動檔的目錄
file_limit=50             # 所能處理的檔案數目上限
...
if [ -d "$startup_dir" ]
then
    # 啟動檔目錄存在，因此可以讀進初始檔。
    echo "initialising file processing..."
```

11.1.2 變數與常數

標頭以及註解只是用來為你的程式碼提供文件的方法之一。另一個方法是使用具描述性的變數名稱。好的變數名稱應該可以呈現變數所代表的意義。`x`、`resn` 或 `procd` 之類的變數名稱，只有在命令稿撰寫當時才具有意義。六個月之後再看到，它們將會變得神秘難解。

好的變數名稱應該簡短而具有描述性。前面所舉的 3 個例子，若能寫成 `file_limit`、`resolution` 和 `was_processed`，將會比較有意義。但不要把名稱弄的太長：像是 `horizontal_resolution_of_the_picture` 這樣的變數名稱只會讓命令稿變得雜亂無章，你反而無法從如此具描述性的變數名稱獲得任何好處。

常數應該以大寫字母來表示，而且通常應該宣告成僅供讀取：

```
declare -r CAPITAL_OF_ENGLAND="London"
```

你應該總是以常數來避免“神奇數字”散落在程式碼各處。例如：

```
if [[ $process_result == 68 ]]
...

```

應該被代換成：

```
declare -ir STAGE_3_FAILURE=68
...
if [[ $process_result == $STAGE_3_FAILURE ]]
...

```

這麼做不僅可以提升程式碼的可讀性，也可以讓值的變更較為容易，尤其是如果此值被多次使用在命令稿中時。

11.2 命令稿的調用

我們在第 6 章曾提到如何使用 `getopts` 來取得傳遞給 `shell` 的選項和引數。這個命令除了可以讓命令稿的設計人員輕易處理使用者所提供的選項和引數，它還可以帶來什麼好處？程式設計人員必須致力於讓使用者的日子儘可能好過一點。最讓使用者生氣的莫過於命令稿：所讀取的不是標準的引數、不提供用法訊息、不以預期的方式處理引數，以及強迫使用者按照程式設計人員自認為對的方式思考。必須檢視原始碼才有辦法找出命令稿接受哪些引數或選項，通常是壓垮駱駝的最後一根稻草！

自由軟體基金會出版了一套撰寫 GNU 軟體的指導方針，其中對 UNIX 公用程式應該如何運作提供了一套標準的做法。【註】當你在撰寫自己的 shell 命令稿時，值得遵照這些指導方針來進行，因為你的命令稿將會讓使用過其他命令列程式的人覺得似曾相識。

你的命令稿至少應該提供單字母選項（像是 `-h`）以及具雙破折號的長選項（像是 `--help`）。它還應該提供兩個選項：`--help` 和 `--version`。以下的內容摘錄自《GNU coding standards》：

--version

這個選項應該會讓程式在標準輸出上印出它的名稱、版本、出處以及版權等資訊，然後順利結束。一旦看到此選項，其餘的選項和引數應該被略過，而且程式不應該執行它的標準功能。

--help

這個選項應該會讓程式在標準輸出上印出如何調用該程式的簡要說明，然後順利結束。一旦看到此選項，其餘的選項和引數應該被略過，而且程式不應該執行它的標準功能。

在 `--help` 選項之輸出的結尾處附近應該有一列說明如何寄送瑕疵報告。它應該具備如下的格式：

```
... Report bugs to mailing-address.
```

表 11-1 列示了若干常用的單字母和長選項，你可能會想要使用在自己的命令稿中。

表 11-1：你可能會用到的選項

| 長選項 | 選項 | 應用在哪些命令上 |
|-----------------------------|-----------------|--|
| <code>-all</code> | <code>-a</code> | <code>du</code> 、 <code>ls</code> 、 <code>nm</code> 、 <code>stty</code> 、 <code>uname</code> 、 <code>unexpand</code> |
| <code>--append</code> | <code>-a</code> | <code>etags</code> 、 <code>tee</code> 、 <code>time</code> |
| <code>--binary</code> | <code>-b</code> | <code>cpio</code> 、 <code>diff</code> |
| <code>--blocks</code> | <code>-b</code> | <code>head</code> 、 <code>tail</code> |
| <code>--date</code> | <code>-d</code> | <code>touch</code> |
| <code>--directory</code> | <code>-d</code> | <code>cpio</code> |
| <code>--exclude-from</code> | <code>-X</code> | <code>tar</code> |
| <code>-file</code> | <code>-f</code> | <code>fgrep</code> |
| <code>--help</code> | <code>-h</code> | <code>man</code> |



註 你可以在 <http://www.gnu.org/prep/standards/> 找到這份文件 — 《GNU coding standards》。

表 11-1：你可能會用到的選項（續）

| 長選項 | 選項 | 應用在哪些命令上 |
|-------------|----|-------------|
| --long | l | ls |
| --line | l | wc |
| --links | L | cpio、ls |
| --output | -o | cc、sort |
| --quiet | -q | who |
| --recursive | r | rm |
| --recursive | R | ls |
| --silent | -s | 與 -quiet 同義 |
| --unique | -u | sort |
| --verbose | -v | cpio、tar |
| --width | -w | pr、diff |

一個能夠讀取一或多個輸入檔並且產生一個輸出檔的命令，就是只以輸入檔為標準引數（亦即 *command filename*）並以一個選項來指定輸出檔（亦即 *command -o filename*）的好典範。

此外，還要注意你的命令稿是否需要用到的使用者所設定的環境變數。如果你的命令稿必須依靠使用者所設定的環境變數，你最好將命令稿重新設計成能夠透過引數傳入此值。

11.3 潛在的問題

這一節將會探討撰寫 shell 命令稿時若干應該注意的事項。有了這方面的認識之後，不僅可以節省你尋找瑕疵的時間，還能夠讓你的命令稿更健全、更具可讀性並且更容易維護。

- 不要建立無所不包的大型命令稿或函式。將程式的機能劃分成較小的單元並把它們把在函式中。這不僅可以讓程式碼較容易閱讀，還可以讓程式碼較容易除錯。
- 總是將 shell 執行指令（例如 `#!/bin/bash`）擺在命令稿的開頭，以確保你的命令稿將會被 *bash* 執行。
- 不要在變數名稱中使用保留字。這麼做將會讓人混淆不清：

```
let let="echo"
let echo="hello"
echo "$echo world"
```

- 注意空白符號。如下的賦值動作將不會得到預期的結果：

```
cat = 5
```

- 不要為變數和函式使用相同的名稱：

```
function letter
{
    echo $letter
}
```

```
letter=letter
letter letter
```

這麼做將會產生混淆不清的結果。欲防範此問題，請試著以動詞來為函式命名，例如 `function print_letter`。

- 測試算符 [...] 的使用要謹慎為之。接下來的兩道 `if` 述句並不一樣，儘管它們看起來非常類似：

```
if [ "$var" = 42 ]
if [ "$var" -eq 42 ]
```

第一道述句所進行的是字串比較，第二道述句所進行的是整數比較。欲進行算術比較，建議在 `if` 述句中使用 `((...))`。

11.4 何時不要使用 `bash`

有些時候，你可能會在著手撰寫命令稿數小時之後，赫然發現自己建立了一個包含數百列複雜程式碼的怪物。儘管這不見得是壞事，不過「總是提醒自己是能夠以更好的方法來完成工作」才是好主意。

通常程式語言的選擇應該在設計階段進行。如果你想在 `UNIX` 系統上從頭開始撰寫程式，你將會有許多選擇，包括 `C` 和 `C++`、`perl`、`python` 以及許多其他的程式語言。這些程式語言各有其優缺點，而且沒有任何一種程式語言可以成為任何問題的最佳解決方案。

如果你發現自己的命令稿需要快速進行大量的處理，或者如果你的命令稿所需要的數學能力不僅是簡單的整數計算，你可以考慮看看是否使用 `C` 或 `C++`。如果你需要的是較好的移植性，`python` 或 `perl` 可能是較佳的選擇。

然而，就算 `bash` 不適合做問題的最終解決方案，你可能會發現 `bash` 是測試你的解決方案以及嘗試各種選項的絕佳語言。