第十章　簡易的 Shell Scripts

10.1 Shell Scripts 及函式簡介

10.1.1 何謂 scripts

當我們想要藉由執行一連串的指令而得到輸出的結果時，就可以將這些程式的組合寫入一個檔案中，這個檔案就叫做 script file。所以一支 Shell Script 就是一個含有 shell 命令組合的檔案，同時也可以算是一支 shell 程式，藉由執行此 script file 就可以得到一連串執行的結果。 當然以上所講的只是一般陽春型的 script file，若您想要讓您所設計的 script 變得比較聰明並具有判斷的能力，也就是在什麼條件下才作什麼事，那麼我們可以在 scripts 裡邊加上一些 if 條件句、for 及 while (until) 迴圈、case 流程控制等等，至於詳細的內容待會都會有範例來作說明。

當我們寫好一支 script 時，必須先讓他擁有執行的權限，這樣您才可以執行它，不過若是您以一般使用者的身分寫一支 shell script，那麼您必須對這支 script 同時擁有 r、x 的權限才行，這點請您多加留意。

從剛剛一直到現在好像都沒提到 " 編譯 " 這兩個字，當然囉，因為 Shell Scripts 是屬於直譯式的語言，所以當您程式碼編寫好後不需經過編譯器的編譯就可以直接執行，很方便吧。

10.1.2 如何撰寫一支 shell script

以下列出要編寫一支 script 的大略內容：

1. 在 script 的一開頭先宣告此 script 預設所使用的 shell 環境為何 ？ 如果是 bash 的話，那就像這樣：

   ```
   #! /bin/bash
   ```

   另外您在執行 file 指令來檢視 script 檔案類型時，就是根據您這裡的宣告噢。
2. 註明此 script 的用途、簡易描述、作者、建檔日期等，好方便管理，而這些內容前面記得加上 "#"，在 "#" 後面的敘述，除了第一行以外，都會被當成註解來看待而不會被執行。
3. 接著就可以開始宣告一些變數及編寫您欲執行的工作內容。
4. 撰寫完畢後，記得將您的 script 改成具有可執行的權限。

5. 為方便直接輸入命令，可以將您 scripts 所在的目錄加入 PATH 環境變數中，或者將所有 scripts 都集中在 ~/bin 裡邊，不過您需先確定 ~/bin 有定義在 PATH 中。

範例：

```
barry@suselinux:~> cd bin
barry@suselinux:~/bin> vi test.sh
#! /bin/bash
# description : test
# author : barry
# date : 2005.10.8
echo "it is my first shell script."
echo -n "my kernel version is : "
uname -r

barry@suselinux:~/bin> chmod +x test.sh

→ 讓 test.sh 具有可執行權限。

barry@suselinux:~/bin> echo $PATH
/home/barry/bin:/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/games:/opt/gnome/bin:
/opt/kde3/bin:/usr/lib/java/jre/bin

→ /home/barry/bin 目錄有存在於 PATH 中，所以等一下就可以直接執行 test.sh。

barry@suselinux:~/bin> test.sh
it is my first shell script.
my kernel version is : 2.6.5-7.97-default

→ 這支 script 的執行結果。
```

如果您的 script 所存在的目錄是在 /tmp 目錄下，由於 /tmp 在 PATH 中並沒有定義，此時您可採取以下的方式來執行這支 script：

法一：

```
suselinux:/tmp # ./test.sh
suselinux:/tmp # /tmp/test.sh
suselinux:/tmp # bash test.sh
```

使用這種方式執行 scripts 時，首先 shell 會產生一個非互動模式的 subshell (shell 的子行程)，再由 subshell 取得 scripts 中所要執行的指令 (subshell 的子行程) 並執行之，等執行完畢後再將主控權交還給原本的 shell。像前一個範例的 script 可以直接被執行時 (script 所在的目錄在 PATH 中有定義)，也是這樣的執行流程。

法二：

```
suselinux:/tmp # .  test.sh
suselinux:/tmp # source test.sh
```

使用 source 或 "." 來執行 scripts 時，會在當前的 shell 下執行此 scripts 中的指令，就好像在命令列上輸入指令一樣。

10.1.3 BASH 函式簡介

若您是第一次學程式語言的話，可能會對這個名詞感覺到陌生，其實說穿了函式只不過是另一個可以被執行的程式罷了，以 script 來說就好像是 script 中的 script 一樣。

函式本身有所謂的函式名稱，我們可以將 shell 程式碼寫在這個函式裡頭，並將其存入 shell 的記憶體之中，以便隨時被呼叫執行。函式在其他程式語言裡又被稱為副程式，這可是程式設計師的最愛，因為使用函式有許多的好處：

1. 簡化程式的設計：
   往往我們在設計一些程式時，可能都會使用到同一段的程式碼，這時候您可以將這些經常使用到的程式碼給它設計成函式，這樣當主程式需要用到這段程式碼時，適時的去呼叫函式就行了，如此便可大大減少了重複寫程式的麻煩。
2. 加快執行效率：
   主程式在引用函式時，是直接於記憶體中作存取，因此可加快函式的執行。
3. 便於維護與修改：
   如果您所設計的程式，其中含有大量的程式碼，這時候我們可以把幾段重要的程式碼給它放到不同的函式裡頭，這樣可以讓主程式的程式碼便得簡潔且易於閱讀，改天您要是想調整或修改這支程式的部分功能時，可以針對函式去做處理即可。

至於宣告函式的方法有以下兩種：

1.
```
function name
{
        指令敘述
}
```

   這裡是使用 function 來宣告函式名稱為 name 的函式，而此函式要執行的工作則指定在 { } 之間。

2.
```
name ( )
{
        指令敘述
}
```

   這是另一種宣告函式的方式，在 name 與括弧之間可以有空白字元，或者也可以連在一起。

您可以在 script 中直接宣告函式，也可以將函式定義在環境設定檔或其他自訂的檔案裡。不過請您注意一點，當您把函式定義於後者時，您所設計的 script 在引用函式之前，需先對存放函式的檔案執行 source 才行，比如 " source /etc/bash.bashrc"，這樣才能先行引入函式。

命令執行的優先順序：

當我們在命令列上輸入一個命令給 shell 解析時，如果同時存在一個別名、函式與指令的名稱都與您所輸入的這個命令名稱相同，則 shell 執行的優先順序為 **別名 → 函式 → shell 內建命令 → 存在於 PATH 所定義的目錄列表中之程式** ，如果沒有別名及函式名稱存在時，才會執行程式本身。底下我們以一個範例來作說明：

先寫一支簡單的 script：

```
suselinux:~/bin # vi barry

#! /bin/bash
# description : test
echo "it is barry's shell script. "
echo "hello ! everybody."
```

```
suselinux:~/bin # chmod +x barry

suselinux:~ # barry
it is barry's shell script.
hello ! everybody.

→ 這是 barry 這支 script 所執行的結果。
```

接著到 /etc/bash.bashrc 裡去宣告函式：

```
suselinux:~ # vi /etc/bash.bashrc



function barry
{
    echo "barry is a function."
}

suselinux:~ # source /etc/bash.bashrc
```

再執行一次 barry：

```
suselinux:~ # barry
barry is a function.

→ 這是函式 barry 的執行結果。
```

最後設定別名：

```
suselinux:~ # vi /etc/bash.bashrc

alias barry='echo barry is a alias.'

suselinux:~ # source /etc/bash.bashrc
```

再次執行 barry：

```
suselinux:~ # barry
barry is a alias.
→ 這是別名 barry 的執行結果。
```

看完上面這個例子，應該就可以明白名稱相同時的執行先後順序了吧！
您可執行 " declare -f " 去查看目前存在有哪些函式，並且會詳細顯示每個函式所定義的內容。比如剛剛我們有定義一個 barry 的函式，此時您就可使用 declare 去查詢一下：

```
suselinux:~ # declare -f | grep barry
barry ( )
        echo "barry is a function."
```

您也可以執行 " unset -f function-name " 來移除函式：

```
suselinux:~ # unset -f barry
```

10.2 shell 特殊的內建變數

10.2.1 位置參數 (positional parameter)

在第三章已經跟各位介紹過環境變數的概念，也就是說當我們在登入系統的時候，會靠這些環境變數來將您個人環境建立起來，而這些環境變數中，有些是定義在環境設定檔裡頭，有些則是由 shell 事先定義好的內建變數，當然還有不少地方也都看得到環境變數的影子啦。不過除此之外，shell 還提供一些比較特殊的內建變數，而這些變數常常在我們編寫 scripts 時會使用到。

假使這些內建變數是以數字來命名的，如 0、1、2、3、... 之類的，就稱其為位置參數 (positional parameter)，那當然欲取得其變數值，就在名稱前面加上個 " $ " 符號，如 $0、$1、$2、$3、... 。 $0 代表程式本身的名稱，$1 代表程式後面所接的第一個引數名稱，$2 代表程式後面所接的第二個引數名稱，餘依此類推。

範例：

```
suselinux:~/bin # vi test1.sh
#! /bin/bash
echo "positional parameter : $1 $2"
echo "the program name is $0"

suselinux:~/bin # chmod +x test1.sh
suselinux:~ # test1.sh para1 para2
positional parameter : para1 para2
the program name is /root/bin/test1.sh
```

上面範例沒問題的話，接著來談函式中的位置參數。由於每個函式都會去處理屬於它自己的引數，因此當您將位置參數設定在函式裡頭的時候，其所取得的將是函式本身的引數，而非命令列上的引數，至於 $0 還是指程式本身。看以下的範例就曉得了：

```
suselinux:~/bin # vi test2.sh
#! /bin/bash
function func
{
      echo '$0' is : $0
      echo '$1' is $1 and '$2' is $2
}

func arg1 arg2

suselinux:~/bin # chmod +x test2.sh
suselinux:~ # test2.sh para1 para2
$0 is : /root/bin/test2.sh
$1 is arg1 and $2 is arg2
```

補充一下，在函式中的程式碼只有在呼叫函式時才會被執行，平常並不會主動執行的。

10.2.2 其他重要的內建變數

另外還有幾個 shell 的內建變數也是您在設計 scripts 時常會引用的：

$#	代表的是位置參數的總數。
$*	代表程式後面的所有位置參數名稱列表。
$@	與 $* 是一樣的意思，但如果使用**雙引號**將它們括住的時候，就有所差異了，這部分在稍後的範例中會說明。
$?	表示程式的結束狀態 (exit status) 傳回值 (exit value 或 return value)。一般傳回 0 表示程式執行無誤 (true)，若傳回非 0 的值，表示程式執行過程出了一點狀況 (false)，至於什麼狀況下傳回什麼值，就要看程式設計者本身的設定了。因此一般程式設計師在寫程式時都會為不同的錯誤訊息設定不同的結束狀態傳回值，如此一來就可以依此傳回值而得知程式執行過程究竟是出了什麼狀況。 在 scripts 中，我們可以使用 **exit** 來退出 script 的執行，並可同時設定 exit value。但如果是在函式裡頭，則需使用 **return** 來結束函式的執行，當然也可同時將 return value 設定上去。

	suselinux:~ # **cat /etc/passwd > bckfile** suselinux:~ # **echo $?** 0 ← 表示 cat 指令有正確被執行。 suselinux:~ # **nocmd** bash: nocmd: command not found suselinux:~ # **echo $?** 127 ← 表示 nocmd 指令執行有誤。	
$$	代表當前 shell 的 pid。	

範例：

```
suselinux:~/bin # vi test3.sh
#! /bin/bash
echo "\$* : positional parameter : $*"
echo "\$@ : positional parameter : $@"
echo Total is : $#
exit 0

suselinux:~/bin # chmod +x test3.sh
suselinux:~/bin # test3.sh para1 para2
$* : positional parameter : para1 para2
$@ : positional parameter : para1 para2
Total is : 2
→ 可以看出 $@ 與 $* 的結果相同。

suselinux:~/bin # echo $?
0
→ exit value 是 0。
```

現在將這 script 內容改一下：

```
suselinux:~/bin # vi test3.sh
#! /bin/bash
echo "\$* : positional parameter : "$*""   ← 將 $* 加上雙引號。
echo "\$@ : positional parameter : "$@""   ← 將 $@ 加上雙引號。
echo Total is : $#
exit 0
```

```
suselinux:~/bin # test3.sh "para1 para2" para3
$* : positional parameter : para1 para2 para3
$@ : positional parameter : para1 para2 para3
Total is : 2

→ 由於 para1 及 para2 被雙引號括住，所以會被當成一個引數來看待，因此 $# 的結果是 2。另外 "$*" 及 "$@" 看起來好像是相同的結果，但實際上 "$*" 代表的是 "para1 para2 para3" 這個單一字串，而 "$@" 則是表示 "para1 para2" 及 "para3" 這兩個字串。
```

如果最後面那個關於 " $* " 及 " $@ " 的範例您還不是很清楚的話，請繼續看下一小節的說明。

10.2.3 "$*" 與 "$@" 的差別

* 與 @ 這兩個內建變數是包含了除位置參數 0 以外的其他位置參數，當您取得它們的變數值時，如果沒有加上雙引號，則兩者的結果是相同的，至於加上雙引號後可就不同了。

"$*" 所代表的是一個包含除位置參數 0 以外的所有位置參數的單一字串，並且以 IFS 中的第一個預設字符來作為每個位置參數之間的分隔符號，而這個字符通常為空白字元，比如當您執行：

```
suselinux:~ # script.sh arg1 arg2 arg3
```

則 "$*" 所代表的就是 " arg1 arg2 arg3 "。

利用這個特性，我們可以藉著改變 IFS 的預設分隔符號而以不同的分隔符號來區隔這些位置參數。

```
suselinux:~/bin # vi test4.sh
#! /bin/bash
IFS= :
echo $*   ← $* 未加雙引號。

suselinux:~/bin # chmod +x test4.sh
suselinux:~/bin # test4.sh arg1 arg2 arg3
arg1 arg2 arg3
```

```
suselinux:~/bin # vi test4.sh
#! /bin/bash
IFS= :
echo "$*"   ← $* 加上了雙引號。

suselinux:~/bin # test4.sh arg1 arg2 arg3
arg1:arg2:arg3

→ 看出加上雙引號之後的結果了吧。
```

<< IFS 及 metacharacter 的說明 >>

IFS 全名為 Internal Field Separator，是 shell 預設所使用的內部欄位分隔符號，這個符號通常是 Space (空白鍵)、Tab (跳位鍵) 及 NewLine (Enter 鍵)，不過 "$*" 真正會引用的分隔符號是以 IFS 第一個定義的字符為主，也就是空格啦。
另外大家常會在 command line 上使用引號，但卻不曉得這些引號的主要作用，因而常搞到暈頭轉向，所以筆者在這裡稍微說明一下，不過在此之前您得先了解 metacharacter 的涵義才行。 metacharacter 是所謂的中介字符，而剛剛才提到的 IFS 字符就是屬於 metacharacter，此外像我們常在 command line 所輸入的一些符號如「 ; 」、「 (」、「) 」、「 < 」、「 > 」、「 & 」、「 | 」等等，也都稱為 metacharacter。當您想把這些符號的功能取消時，就可以使用單引號 (single quotes) 或雙引號 (double quotes) 或 「 \ 」(backslash)。

當您使用單引號時，以上所列 metacharacter 的功能都會被取消，另外像別名、「 ~ 」、「 $ 」「 ` 」(這是反引號而非單引號) 及萬用字元等，其功能也都會被關閉。

當您使用雙引號時，除了「 $ 」、「 ` 」 能保留原有的功能以外，其他剛剛所提及的也都會被關閉。

再來就是使用「 \ 」時，只有緊接在「 \ 」之後的特殊字符才會取消其功能。

看完以上說明，可能心理還有些問號。記不記得在介紹 grep 及 sed 指令時，後面的 pattern 都用單引號包住，但裡頭的一些特殊字符還是有其功能存在啊。關於這個問題必須從 shell 及 command meta 的角度來看，首先當您執行一些支援 RE 的指令時，指令後之所以加上單引號是為了避免這些特殊字符提早被 shell 解析掉，當使用單引號包住後就可以把那些字符完整的交給支援 RE 的 command 來處理，那當然此時這些特殊符號就能代表在 RE 中的那些特殊涵義囉 ！

了解了 "$*" 後，接著來看 "$@"。"$@" 可以將每個位置參數用雙引號括住，這樣做的目的是可以讓您個別取用它們的值來使用：

```
suselinux:~ # script.sh arg1 arg2 arg3
```

此時 "$@" 所代表的就是 "arg1" "arg2" "arg3" 這三個字串。

範例說明：

```
suselinux:~/bin # vi test5.sh
#! /bin/bash
function arg {
        echo "use double quotes :"
        echo -e "\tTotal is $#"
}
arg "$*"

suselinux:~/bin # chmod +x test5.sh
suselinux:~/bin # test5.sh arg1 arg2 arg3
use double quotes :
        Total is 1

→ 因 "$*" 代表 "arg1 arg2 arg3"，所以 $# 是 1。

suselinux:~/bin # vi test5.sh
#! /bin/bash
function arg {
        echo "use double quotes :"
        echo -e "\tTotal is $#"
}
arg "$@"

suselinux:~/bin # test5.sh arg1 arg2 arg3
use double quotes :
        Total is 3

→ 因 "$@" 代表 "arg1" "arg2" "arg3"，所以 $# 是 3。
```

再舉個更簡單的例子，不過因為 script 中會使用到 for 迴圈，所以這個範例可等您熟悉 for 迴圈的用法後，再回過頭來看：

```
suselinux:~/bin # vi test6.sh
#! /bin/bash
for var in "$*"
do
   echo $var is positional parameter.
done

suselinux:~/bin # chmod +x test6.sh
suselinux:~/bin # test6.sh arg1 arg2 arg3
arg1 arg2 arg3 is positional parameter.

suselinux:~/bin # vi test6.sh
#! /bin/bash
for var in "$@"
do
   echo $var is a positional parameter.
done

suselinux:~/bin # test6.sh arg1 arg2 arg3
arg1 is a positional parameter.
arg2 is a positional parameter.
arg3 is a positional parameter.
```

10.3 test 指令的用法

在 10.2.2 節中提到了程式結束狀態的傳回值 (exit value)，如 exit value 為 0 就是 true，exit value 非 0 即是 false。而我們現在要介紹的 test 指令則是可以針對您所提供的條件來作測試，當條件測試結果為 true 時，test 就傳回 0，測試結果為 false，那當然就傳回非 0 的值。在 scripts 中，test 可是很常用的一個測試工具，所以請您務必了解。

10.3.1 test 指令

指令語法：test　EXPRESSION
　　[EXPRESSION]

EXPRESSION 就是您所輸入的條件，而 test 就是根據這個條件來決定要傳回什麼值。您也可以使用 [ ] 來代替 test 的執行，不過包在 [ ] 中的 EXPRESSION 可要與 [ ] 之間保有空白字元噢，也就是在 "[" 之後及 "]" 之前要有空格啦。

10.3.2 運算符

我們在設計 EXPRESSION (條件式) 的時候，常常會用到一些運算符號於其中，然後再交給 test 去做判斷，所以底下就分別針對字串、數值及檢查檔案屬性的判斷運算符來跟大家作說明。

檔案屬性的運算符：

-f file	file 存在，並且為一檔案。
-d file	file 存在，並且為一目錄。
-r file	執行此 script 者對 file 具有讀取權限。
-w file	執行此 script 者對 file 具有寫入權限。
-x file	執行此 script 者對 file 具有執行權限。
-e file	file 存在於系統上。
-s file	file 存在，並且大小不為 0。
-u file	file 具有 SUID 的屬性。
-g file	file 具有 SGID 的屬性。
-k file	file 具有 Sticky bit 的屬性。
file1 -nt file2	file1 較 file2 為新 (根據修改時間作比較)。
file1 -ot file2	file1 較 file2 為舊 (根據修改時間作比較)。

您可以加上 "!" (not) 符號在運算符之前，比如 "! -d file" 就是說 file 不是一個目錄。

數值比較的運算符：

n1 -eq n2	n1 等於 n2。-eq : equal。
n1 -ne n2	n1 不等於 n2。-ne : not equal。
n1 -gt n2	n1 大於 n2。-gt : greater than。
n1 -ge n2	n1 大於等於 n2。-ge : greater than or equal。
n1 -lt n2	n1 小於 n2。-lt : less than。
n1 -le n2	n1 小於等於 n2。-le : less than or equal。

字串比較的運算符：

str1 = str2	str1 等於 str2。
str1 != str2	str1 不等於 str2。
str1 > str2	str1 大於 str2。
str1 < str2	str1 小於 str2。
-z str	str 為空字串。
-n str	str 為非空字串。

10.3.3 " || " 及 " && " 邏輯算符

在 test 語法中也可以使用 " || " 及 " && "。還記得我們在第三章提到的 "command1 && command2" 及 "command1 || command2" 的意思吧！不過那時候並沒有用傳回值的觀念來解釋。先說 "command1 && command2"，當 command1 的 exit value 為 0 (true) 時才會接著執行 command2，若 command 的 exit value 為非 0 (false)，則 command2 將不會被執行，也就是說 command2 只有在 command1 為 true 的情況下才會被執行。而 "command1 || command2" 則是表示 command2 只有在 command1 為 false 時才會被執行囉！有了這個觀念後，那 test 也是一樣的意思，比如：

```
test expression && command
```

這表示說當 test 測試結果傳回 0 時，就會執行 command，那如果使用 " || " 時，則只有在 test 傳回非 0 時才會執行 command。

在 expression 中您也可以使用 -a (and) 及 -o (or) 來對一個以上的條件做判斷，
以下舉幾個例子來看看如何使用 test (或 [ ])並搭配運算符使用：

```
suselinux:~ # test -f /etc/passwd && echo "passwd is a file."
passwd is a file.

→ 測試 /etc/passwd 是否為一般檔案，如果是的話 (ture)，test 就傳回 0，並接著執行其後的 echo
    指令。

suselinux:~ # [ -f /etc/passwd ] && echo "passwd is a file."
passwd is a file.
→ 以 [ ] 來替代 test。這種用法也要熟悉噢。
```

```
suselinux:~ # test ! -x /usr/bin/passwd || var="The expression is false."
suselinux:~ # echo $var
The expression is false.
```

→ 判斷 /usr/bin/passwd 是否不可執行，結果當然是 false，所以接著就把 "The expression is false." 這個值設定給 var 變數。

```
suselinux:~ # var1=3;var2=5
suselinux:~ # [ "$var1" -lt "$var2" ] && [ ! -s /etc/passwd ] || echo "good"
good
```

→ 第一個 expression 為 true，所以會進行第二個 expression 的判斷，如判斷結果為 false 才會執行 echo 指令。

```
suselinux:~ # [ -n "$USER" -a "$UID" = 0 ] && echo "you are super user."
you are super user.
```

→ 這個條件句中，使用 -a 來連接兩個條件的判斷，當兩條件都正確時，這個條件句才是 true，也才能繼續進行下面的 echo 指令。

10.4 if 條件判斷式 (if --- then --- elif --- then --- else --- fi)

if 條件句應該算是程式語言裡使用率極高的一種條件判斷式，它可以根據您所設定條件句的真假來決定後續要採取的行為。先來了解它的基本用法：

```
if [ 條件句 1 ] ; then
      指令敘述 1
elif [ 條件句 2 ] ; then
      指令敘述 2
else
       指令敘述 3
fi
```

上面意思是說如果條件句 1 測試結果為真 (傳回 0)，則執行 then 後面的指令敘述 1，執行完畢後整個 if 述句就結束。但如果條件句 1 為假 (傳回非 0) 時，則再進行 elif 後另一個條件句 2 的判斷，此時若條件句 2 判斷結果為真，就執行指令敘述 2，如果條件句 2 判斷亦為假時，則執行 else 其後所指定的指令敘述 3。

在 if 條件句中，您可以使用好幾個 elif 去額外指定一些不同的條件。在 if 流程裡，elif 及 else 並非必要，但是 then 則是一定要的。

當然 if 條件句的語法 **if [...]** 也可以使用 **if test ...** 來替代，請參考上一節 test 的相關說明噢。

使用 if 來做複合式的判斷也很簡單，就是使用剛剛才教過的 " && " (and)、" || " (or) 來連接兩個條件句，比如：

```
if [ 條件句 1 ] && [ 條件句 2 ] ; then
    指令敘述 1
else
    指令敘述 2
fi
```

這是說當兩個條件式皆為真的情況下才會執行指令敘述 1，否則就執行指令敘述 2。若換成 " || " 時，則只要條件句 1 或條件句 2 有一者恆為真，就會執行指令敘述 1，只有當兩者皆為假時才會執行指令敘述 2。

如果 if 後面接的是 command，比如：

```
if command1 && command2 ; then
    指令敘述 1
else
    指令敘述 2
fi
```

這表示說當 command1 傳回 true 時才會接著執行 command2，若 command2 也傳回 true，那就執行指令敘述 1，否則執行指令敘述 2；但若 command1 傳回 false，那 command2 根本就不會被執行，自然就只能執行 else 後的指令敘述 2 了。

您亦可對條件句與 comand 做複合式的判斷：

```
if [ 條件句 1 ] || command1 ; then
    指令敘述 1
else
    指令敘述 2
fi
```

當條件句 1 為 true 或者條件句 1 為 false 但 command1 為 true 時，會執行指令敘述 1，否則執行指令敘述 2。

範例：

```
suselinux:~/bin # vi script1.sh
#!/bin/bash
var=/etc/passwd
if test -e "$var" && grep -q barry "$var" ; then
        echo "`id barry`"
else
        echo "passwd file is not exist or barry not found."
fi

suselinux:~/bin # chmod +x script1.sh
suselinux:~/bin # script1.sh
uid=1008(barry) gid=100(users)
groups=100(users),14(uucp),16(dialout),17(audio),33(video)
```

[-e "$var"] 測試結果為 true，所以接著執行 grep 指令，如果此時 /etc/passwd 裡存在著 barry 這個帳號的話，那麼 grep 也會傳回 true，最後當然就執行 then 後面的指令敘述了。

範例：

```
suselinux:~/bin # vi script2.sh
#! /bin/bash
a=3
b=5

if [ "$a" -gt 6 -o "$a" -le "$b" ] ; then
        echo the larger number is "$b".
else
        echo it is bad .
fi

suselinux:~/bin # chmod +x script2.sh
suselinux:~/bin # script2.sh
the larger number is 5.
```

這是一個 expression 裡邊含有兩個條件的測試，因為兩條件是使用 -o 做連接，所以只要其中一個條件正確，則其結果就恆為真。

範例：

```
suselinux:~/bin # vi script3.sh
#! /bin/bash
prog=/usr/sbin/named
if [ -e "$prog" ] && [ -n "$(/sbin/pidof named)" ] ; then
        echo named is running now.
else
        echo named is not running.
fi

suselinux:~/bin # chmod +x script3.sh
suselinux:~/bin # script3.sh
named is not running.

suselinux:~/bin # rcnamed start   ← 將 named 服務啟動後再跑一次 script ，並與剛剛執行的結果比較一下。

suselinux:~/bin # script3.sh
named is running now.
```

首先 /usr/sbin/named 是存在的，所以 [-e "$prog"] 為真 ； 而 /sbin/pidof named 是用來查詢 named 的 pid，假設 named 服務根本沒有啟動，則執行 /sbin/pidof named 就無法顯示任何 pid 的資訊，當然就算是空字串了，所以 [-n "$(/sbin/pidof named)"] 為假，因此會執行 else 後面的指令敘述而出現 "named is not running" 的執行結果出來。

範例：

```
suselinux:~/bin # vi /root/testfile
var1="`grep -w barry /etc/passwd | cut -d : -f 3`"
var2="`grep -w barry /etc/passwd | cut -d : -f 4`"

suselinux:~/bin # vi script4.sh
#! /bin/bash
[ -e /etc/passwd ] || exit 5   ← 如果 /etc/passwd 不存在時，直接退出此 script 的執行，並設定其 exit value 為 5。
```

```
if [ -e /root/testfile ] ; then    ← 如果 /root/testfile 存在，則將 var1 及 var2 讀入記憶體，以隨時
                                     被取用，否則就結束 script 的執行。
    .  /root/testfile
else
    exit 6
fi

if [ -n "$var1" -a ! -z "$var2" ] ; then    ← 如果 $var1 及 $var2 皆非空字串，則執行以下
                                              兩個 echo 指令。
    echo "barry's uid is $var1"
    echo "barry's gid is $var2"
else
    echo "barry's id not found."    ← 上面測試為假時，則執行此處的 echo 指令，執行完畢
                                      便結束 script 的執行。
    exit 7
fi

if [ "$#" -eq 0 ] ; then    ← 如果執行此 script 時，未接任何引數則顯示 "Success"，否則會提示
                              您不需輸入位置參數。
  echo Success.
else
  echo You don't need to input positional parameter.
  exit 8
fi

suselinux:~/bin # chmod +x script4.sh
suselinux:~/bin # script4.sh
barry's uid is 1008
barry's gid is 100
Success.

suselinux:~/bin # script4.sh arg
barry's uid is 1008
barry's gid is 100
You don't need to input positional parameter.

suselinux:~/bin # echo $?
8
```

10.5 數值運算及 shift 的使用

10.5.1 數值運算

當我們在宣告變數時，若無特別指定，會將變數的內容當成字串來看待，那萬一我們想做數值運算時怎麼辦呢 ？ 以下提供幾種做法：

- 使用 " declare -i " 來宣告此變數為整數變數。
- 使用 expr 這個外部指令來做數值運算。
- 使用 let 這個 bash 的內建命令來做數值運算。

至於數值運算的加減乘除就使用 「+」、「-」、「*」、「/」 就行了。

範例：

```
suselinux:~/bin # vi script5.sh
#! /bin/bash
a=6
b=3
declare -i c=$a-$b   ← 宣告變數 c 為整數。
let d=$a*$b   ← 使用 let 指令做數值運算。
e=`expr $a / $b`   ← 使用 expr 指令做數值運算。
f=$(expr $a + $b)   ← 與 f=`expr $a + $b` 是一樣的。
g=$(($a+$b))   ← 另一種數值運算的方式。

if [ "$c" -eq 3 -a "$d" -eq 18 -a "$e" -eq 2 -a "$f" -eq 9 -a "$g" -eq 9 ] ; then
      echo "c=$c ; d=$d ; e=$e ; f=$f ; g=$g"
else
      exit 3
fi

suselinux:~/bin # chmod +x script5.sh
suselinux:~/bin # script5.sh
c=3 ; d=18 ; e=2 ; f=9 ; g=9
```

如果條件句測試結果為真，則執行 then 後面的指令敘述，否則執行 exit 並設定其 exit value 為 3。

10.5.2 shift 的用法

應該還記得前面所學過的位置參數吧！不過當時您心中或許會有個疑問，當我在 command line 執行類似於 " test.sh -a arg1 arg2 " 時，那 $1 就是 -a，$2 及 $3 才是 arg1 及 arg2，如果現在想要把 $1 對應到 arg1 及 $2 對應到 arg2 可行嗎？當然是可以的啦，只要用 **shift** 就可以搞定了。當我們使用 shift 而未指定 number 時，就是使用預設的 shift 1，如此就可以將 $1 對應到第二個參數，$2 對應到第三個參數，若您使用 shift 2 就可以將 $1 對應到第三個參數，$2 對應到第四個參數，舉例如下：

範例：

```
suselinux:~/bin # vi script6.sh
#! /bin/bash
echo The total of positional parameter is : "$#"
if [ "$1" = -a ] ; then
      echo correct positional parameter :
      shift
      echo -e "\tThe first parameter is $1"   ← 由於前面已執行 shift，所以 1=$2，那 $1 當然
                                                就是指第二個位置參數囉。
      echo -e "\tThe second parameter is $2"   ← 這個應該不用多做解釋了吧！
      echo The total of positional parameter is : "$#"
else
      echo You need to use \"-a\".
fi

suselinux:~/bin # chmod +x script6.sh

suselinux:~/bin # script6.sh -a arg1 arg2
The total of positional parameter is : 3
correct positional parameter :
      The first parameter is arg1
      The second parameter is arg2
The total of positional parameter is : 2

suselinux:~/bin # script6.sh -b arg1 arg2
The total of positional parameter is : 3
You need to use "-a".
```

10.6 迴圈介紹

所謂的迴圈 (loop) 就是指在所指定的條件下而能被重複執行的一段程式碼。在 script 中常使用到的迴圈結構有 for、while 及 until 這三種，請接著看以下小節的說明。

10.6.1 for 迴圈

for 迴圈語法為：**for … in … do … done**

```
for   變數名稱   in   變數值 1   變數值 2   變數值 3 …
do

      指令敘述

done
```

對 for 迴圈來說，若 in 後面有三個變數值，就會執行三次 do、done 之間的指令敘述，比如：

```
for var in barry mary
do
      echo hello,"$var"
done
```

這個意思就是說當 var=barry 時執行 echo 指令，當 var=mary 時再執行一次，總共會執行兩次迴圈，執行完畢就結束。所以執行此 script 的結果，您將能在螢幕上看到 " hello,barry " 及 " hello,mary "。

範例：

```
suselinux:~/bin # vi script7.sh
#! /bin/bash
for x in 5 10 15 20
do
      if [ "$x" -lt 18 ] ; then
            echo The value of x is "$x"
      else
            echo The value of x is too big.
      fi
```

```
        sleep 2   ← if 條件句執行完後，會先延遲兩秒鐘，然後再進行下一個 loop。
done

suselinux:~/bin # chmod +x script7.sh
suselinux:~/bin # script7.sh
The value of x is 5
The value of x is 10
The value of x is 15
The value of x is too big.
```

在 for 迴圈語法中，如果省略了 in 及其後的變數值列表 (list)，則會自動幫您指定成 in "$@"。我們把剛剛那個範例改一下：

```
suselinux:~/bin # vi script7.sh
#! /bin/bash
for x   ← 就同等於 for x in "$@"。
do
        if [ "$x" -lt 18 ] ; then
                echo The value of x is "$x"
        else
                echo The value of x is too big.
        fi

        sleep 2
done

suselinux:~/bin # script7.sh 5 15 25
The value of x is 5
The value of x is 15
The value of x is too big.
```

另外一種 for 的表示法如下：

```
for (( 起始值 ; 條件式 ; 步進式 ))
do
        指令敘述
done
```

範例：

```
suselinux:~/bin # vi script8.sh
#! /bin/bash
b=5
for (( a=0 ; a<=b ; a=a+1 ))
do
      echo The number is "$a".
done
```

→ 當 a=0 時，進行 a<=b 的判斷，如果條件符合則執行迴圈，執行完畢後回到 for 步進式那裡，此時 a 會變成 1，然後再看看 a<=b 是否正確，如果還是符合條件的話，再執行迴圈，一直進行到條件不符合為止。

```
suselinux:~/bin # chmod +x script8.sh
suselinux:~/bin # script8.sh
The number is 0.
The number is 1.
The number is 2.
The number is 3.
The number is 4.
The number is 5.
```

continue 與 break 指令：

這兩個指令常使用在 do --- done 的迴圈裡頭。 continue 是繼續進行下一個迴圈，也就是說在 continue 與 done 之間的指令敘述將不再被執行，而是回到迴圈頂端來準備執行下一次的迴圈。至於 break 則表示要中斷迴圈，也就是迴圈不再被執行了。而 break 與 exit 及 return，一般人很容易搞混，簡單的說，**exit 是用來退出 script 的執行，return 是退出函式的執行，break 則是使用在中斷迴圈上頭**。

範例：請先在 /tmp 目錄中建立兩個大小不為 0 的檔案 file1 及 file2，然後再建立兩個空檔案 file3 及 file4，接著編輯 script：

```
suselinux:~/bin # vi script9.sh
#!/bin/bash
for file in $(ls /tmp/file*)
do
      if [ -f "$file" ] ; then
            if [ -s "$file" ] ; then
```

```
                echo "$file" is not empty.
                continue   ← 在 continue 至 done 中間的指令不會被執行，而是回到 for 來帶入
                             另一個 file 的值，以繼續執行迴圈。
            fi
            echo Delete "$file"
            rm -f "$file"
        fi
done

suselinux:~/bin # chmod +x script9.sh
suselinux:~/bin # script9.sh
/tmp/file1 is not empty.
/tmp/file2 is not empty.
Delete /tmp/file3
Delete /tmp/file4
```

10.6.2 while、until 迴圈

使用 while 迴圈時，一般是根據其後所指定的條件句或指令來做判斷，只要測試結果其 exit value 傳回 0 (true) 就執行迴圈，執行完畢後再回到 while 來重新進行判定，如果還是 ture，那就繼續執行迴圈囉，一直要到傳回非 0 (false) 才會停止。

while 迴圈語法為 while --- do --- done

```
while [ 條件句 ]  (或者是 while command)
do
    指令敘述
done
```

範例：

```
suselinux:~/bin # vi script10.sh
#! /bin/bash
declare -i x=0
while [ "$x" -le 5 ]
do
    echo The value of x is : "$x"
    x=$x+1
    sleep 1
```

```
done

suselinux:~/bin # chmod +x script10.sh
suselinux:~/bin # script10.sh
The value of x is : 0
The value of x is : 1
The value of x is : 2
The value of x is : 3
The value of x is : 4
The value of x is : 5
```

另外一種與 while 迴圈相反的是 until 迴圈，也就是說只有當條件句或指令為假時，才會執行迴圈，一直要到條件句或指令判斷為真時才結束迴圈的執行。

範例：

```
suselinux:~/bin # vi script11.sh
#! /bin/bash
num=0
until [ "$num" -eq 30 ]
do
        echo -n "Please input a number here : "
        read num

        if [ "$num" -gt 30 ] ; then
            echo "$num" is too big , try again.
            echo
        elif [ "$num" -eq 30 ] ; then
            echo BINGO !! you got it.
        else
            echo "$num" is too small , try again.
            echo
        fi
done

→ 這是一支猜數字的 script，應該都看得懂吧 !

suselinux:~/bin # script11.sh
Please input a number here : 50   ← 輸入 50。
```

```
50 is too big , try again.

Please input a number here : 20   ← 輸入 20。
20 is too small , try again.

Please input a number here : 30   ← 輸入 30。
BINGO !! you got it.
```

10.7 case 流程控制

10.7.1 case 的基本語法

先認識一下 case 的語法：

```
case  string  in
      樣式 1 ) 指令敘述 1;;
      樣式 2 ) 指令敘述 2;;
      樣式 3 ) 指令敘述 3;;
      * ) 指令敘述 4;;
esac
```

簡單的說，當 case 後的 string 符合 in 下面所指定的其中一個樣式時，則執行該樣式後的指令敘述，但如果都沒有符合的樣式時，則執行 "*" 後面的指令敘述。語法中要注意的是每個樣式後的最後一個指令輸入完後需使用雙分號 (;;) 來代表指令段落的結束 ; 另外 case 流程的末端必須以 esac 來作為結束，就是 case 倒過來寫啦。

在語法中的 string 部分，一般我們都會使用 "$變數名稱 " 來取得不同的 string，然後再去跟底下的樣式做比對，這樣才能付于 case 流程具有更大的彈性。

在 Linux 裡邊，很多系統服務的 scripts 都是使用 case 流程來控制的，比方我想管理 atd 服務時可執行如下：

```
suselinux:~ # /etc/init.d/atd stop
suselinux:~ # /etc/init.d/atd start
suselinux:~ # /etc/init.d/atd restart
suselinux:~ # /etc/init.d/atd status
```

其實之所以能夠讓您如此方便的管理服務，靠的就是該 script 裡的 case 來控制的，只要我們對 atd script 做如下的設計：

```
case $1 in
        start )
                command1
                command2
                ;;
        stop )
                command3
                ;;
        restart )
                command4
                command5
                ;;
        status )
                command6
                ;;
```

這樣當我們 atd script 後所指定的第一個位置參數為 start 時，自然就會去執行 command1 及 command2，若為 stop 則執行 command3。這樣應該就可以理解 case 好用的地方吧 ！

10.7.2 read 指令介紹

像剛剛那個範例是藉由 command line 所輸入的位置參數來當成 $1 的變數值，而另外一種取得變數值得方式則可透過 **read** 指令來讓您從 keyboard 所輸入的字串當成變數值，所以接著來介紹 read 的簡單用法。

- **read**

  read 指令主要是從 stdin 讀取資料，一般最常見的就是從 keyboard 所做的輸入，然後將此輸入值放進 read 後所指定的變數中，因此 read 指令後面可要指定一個變數名稱噢。

```
suselinux:~ # read var
/etc/passwd ← 這是從鍵盤所做的輸入，完畢後按下 Enter 鍵就會把 /etc/passwd 設定給 var
              這個變數。
```

```
suselinux:~ # echo $var ← 顯示 var 的變數值。
/etc/passwd

suselinux:~ # read -p "Please input your name: " name   ← " -p " 是用來讓您在輸入字串
                                                          前所能看到的提示內容。

Please input your name: barry ← 輸入 barry 並按下 Enter 後，就把這個變數值設定給
                                name 變數。

suselinux:~ # echo $name
barry
```

範例：

```
suselinux:~/bin # vi script12.sh
#!/bin/bash
echo Can you tell me your name ?
echo -n "Please input yes or no: "
read ANS
case "$ANS" in
      [Yy] | yes | YES )
                  read -p "Your name is: " name
                  echo Hello,"$name",thanks for your answer.
                  ;;
      [Nn] | no | NO )
                  echo "It's ok. Good-bye."
                  ;;
      * )
                  echo See you next time.
                  ;;
esac

suselinux:~/bin # chmod +x script12.sh
suselinux:~/bin # script12.sh
Can you tell me your name ?
Please input yes or no: yes ← 這裡可以輸入 yes、YES、y 或 Y。
Your name is: barry
Hello,barry,thanks for your answer.
```

這個範例中的樣式部分使用了 "|"，像 yes|YES 就是比對 $ANS 是否符合 yes 或 YES 樣式，而 [Yy] 則是比對 $ANS 是否符合 Y 或 y 的樣式，因此 [Yy]|yes|YES 組合起來應該就知道意思了吧 ！

10.7.3 加入函式 (function) 的功能

函式的概念在本章一開始已提及，有點遺忘的話，要趕快回過頭去複習一下。如果您將函式寫在 script 裡頭，可要注意到一件事情，就是您所宣告的函式必須設定在 script 的前頭，這樣在執行此 script 時才能先將函式讀入記憶體中，以等待隨時被呼叫。您不用擔心寫在前面的函式裡頭的程式碼會馬上被執行，因沒有人去呼叫它時是不會主動執行的。

範例：

```
suselinux:~/bin # vi script13.sh
#!/bin/bash
user ( )
{
      echo "preparing to add a new user."
      sleep 1
      if [ $(grep "^$USER" /etc/passwd | cut -d : -f 3) -ne 0 ] ; then
            echo you are not administrators , so can not add a new user."
            return 5
      else
            useradd
      fi
}

echo 1. add a new user.
echo 2. exit the script.
echo -n "Enter your choice : "
read choice
case $choice in
       1 ) user ;;
       2 ) exit ;;
       * ) echo "your choice must be 1 or 2." ;;
esac

suselinux:~/bin # chmod +x script13.sh
suselinux:~/bin # script13.sh
```

```
1. add a new user.
2. exit the script.
Enter your choice : 2 ← 輸入 2 會直接退出 script。
suselinux:~/bin #

suselinux:~/bin # script13.sh
1. add a new user.
2. exit the script.
Enter your choice : 1 ← 輸入 1 則會呼叫 user 函式。
preparing to add a new user.
useradd: Too few arguments. ← 這是單純執行 useradd 指令所看到的訊息。
Try `useradd --help' or `useradd --usage' for more information.

suselinux:~/bin # cp script13.sh ~barry/bin
suselinux:~/bin # su - barry
barry@suselinux:~> script13.sh
1. add a new user.
2. exit the script.
Enter your choice : 1
preparing to add a new user.
you are not administrators , so can not add a new user.
```

針對 user 函式中的 if 條件句來稍微解釋一下。在 " grep "^$USER" /etc/passwd " 指令敘述中的 ^$USER 千萬不能用單引號，不然 $ 會被當成一般字元來處理，而造成無法取得 USER 的變數值。另外如果您是以 root 身分執行這支 script，則 $USER 為 root，換成以 barry 使用者來執行的話，其 $USER 為 barry。

10.8 變數值替換及變數值樣式比對

一般狀況下，您使用 " $variable " 及 " ${variable} " 這兩種方式所取得的變數值是相同的，也就是說 " $variable " 只是 " ${variable} " 的一種簡化語法罷了！至於加上 { } 的用意是為了能讓您在 { } 中使用字串運算符來對您的變數值做靈活的操作。

```
suselinux:~ # var=Hello
suselinux:~ # echo $var
Hello
suselinux:~ # echo ${var}
Hello
```

10.8.1 變數值替換

以下我們就來說說幾種變數值替換方面的字串運算符表示方法：

<table>
<tr><td>${var:-str}</td><td>如果 var 變數存在且為非空值，就傳回其值；如果 var 變數存在但為空值或者 var 變數根本不存在，則傳回 str。
suselinux:~ # unset var
suselinux:~ # echo ${var:-/etc/passwd}
/etc/passwd
suselinux:~ # echo $var

補充：
˙" unset var " 是取消 var 變數，也就是 var 已被移除而不存在。
˙" var= " 是將 var 設定成空值 (null value)。</td></tr>
<tr><td>${var:=str}</td><td>如果 var 變數存在且為非空值，就傳回其值；如果 var 變數存在但為空值或者 var 變數根本不存在，就把 str 設定給 var ，並傳回 str。
suselinux:~ # echo $var

suselinux:~ # echo ${var:=/etc/passwd}
/etc/passwd
suselinux:~ # echo $var
/etc/passwd</td></tr>
<tr><td>${var:?info}</td><td>如果 var 變數存在且為非空值，就傳回其值；如果 var 變數存在但為空值或者 var 變數根本不存在，就會顯示 var: 及 info 的訊息。
suselinux:~ # unset var
suselinux:~ # echo ${var:?var is not exist.}
-bash: var: var is not exist.</td></tr>
<tr><td>${var:+str}</td><td>如果 var 變數存在且為非空值，則傳回 str；如果 var 變數存在但為空值或者 var 變數根本不存在，就傳回空值。
suselinux:~ # var=/etc/passwd
suselinux:~ # echo ${var:+test}
test
suselinux:~ # var=
suselinux:~ # echo ${var:+test}</td></tr>
<tr><td>${var:number}</td><td>number 為一數字，若 var 的變數值含有 8 個字元，當 number 為 3</td></tr>
</table>

	時，表示取得第三個字元以後的部分。 suselinux:~ # var=myshellscript suselinux:~ # echo ${var:2} shellscript
${var:n1:n2}	若 var 的變數值含有 8 個字元，當 n1 為 3，n2 為 6 時，表示取得第三個字元以後的六個字元。 suselinux:~ # var=myshellscript suselinux:~ # echo ${var:2:5} shell

10.8.2 變數值樣式比對

另外一種字串運算符的型態是把所指定的樣式（pattern）跟變數值作比對，然後才根據比對的結果來決定傳回的變數值。

${var#pattern}	把 pattern 與 var 變數值做比對時，是從 var 變數值的前面部分比對起，如果有符合之處，就刪除變數值中最短的部分，並傳回其餘的變數值。 suselinux:~ # var=/home/barry/dir/file suselinux:~ # echo ${var#/*/} barry/dir/file
${var##pattern}	把 pattern 與 var 變數值做比對時，是從 var 變數值的前面部分比對起，如果有符合之處，就刪除變數值中最長的部分，並傳回其餘的變數值。 suselinux:~ # var=/home/barry/dir/file suselinux:~ # echo ${var##/*/} file
${var%pattern}	把 pattern 與 var 變數值做比對時，是從 var 變數值的後面部分比對起，如果有符合之處，就刪除變數值中最短的部分，並傳回其餘的變數值。 suselinux:~ # var=/home/barry/dir/ suselinux:~ # echo ${var%/*/} /home/barry
${var%%pattern}	把 pattern 與 var 變數值做比對時，是從 var 變數值的後面部分比對起，如果有符合之處，就刪除變數值中最長的部分，並傳回其餘的

	變數值。 suselinux:~ # var=/home/barry/test.tar.gz suselinux:~ # echo ${var%%.*} /home/barry/test
${var/pat1/pat2}	若變數值中含有 pat1，就將第一個 pat1 用 pat2 作替代 。 suselinux:~ # var=/home/barry/barrydir/barryfile suselinux:~ # echo ${var/barry*/mary} /home/mary suselinux:~ # echo ${var/barry/mary} /home/mary/barrydir/barryfile
${var//pat1/pat2}	若變數值中含有 pat1，就將全部的 pat1 用 pat2 作替代 。 suselinux:~ # var=/home/barry/barrydir/barryfile suselinux:~ # echo ${var//barry/mary} /home/mary/marydir/maryfile