# 第六單元

# Bash Script

## 6.1 Linux 的登入

Linux 是一個多人多工的系統，顧及安全性，每個人都要有一個帳號跟密碼，要打入帳號及密碼『登入』系統以後才能使用，不過我們目前是第一次起動系統，還沒有建立任何帳號，只能以系統預設的 "root" 帳號登入。

跟之前說過的一樣，若你的 X 視窗有設定好的話，並且你也有選擇是否在開機完成後直接進入 X 視窗，這時開機就有類似的畫面，這就是所謂的簽入畫面，如果沒有設定 X 的話，就會用文字的方式表示，這也是最早且最基本的樣式，如下：

```
Red Hat Linux release 6.0    + CLE v0.8
Kernel 2.2.5-15CLE on an i686
Login：
```

RedHat Linux 系統預設提供六個文字介面的虛擬主控台供使用者使用，您可以在不同的主控台登入，同時作不同的事，不會相互干擾，彼此可同時按【Ctrl】+【Alt】+【F1】~【F6】鍵來相互切換主控台，假如您的 X 順利起動了，那麼 X 會佔在第七號主控台，按【Ctrl】+【Alt】+【F7】就可以切換到 X 的畫面，所以有順利起動 X 的人只要按【Ctrl】+【Alt】+【F7】就可以回到 X 了。

當然，不管你是文字或視窗都會有 Login：的字樣，圖形介面的話之前已經介紹過如何簽入，如果是文字視窗的話，這時因為我們是第一次進入系統所以請鍵入 **root** 按【Enter】後，便又出現 Password 的字樣，這時請你再鍵入你之前安裝時所設定的密碼即可，打錯的話，顯示 **login incorrect** 的字樣，這時只好請你重新 login 了！你要是你忘了密碼！這下子可麻煩了！請翻到 FAQ 去求救吧！簽入成功後，便會進入自己帳號的家目錄(家目錄的詳細說明在下一章)，剛開始用 root 帳號進入的朋友，若是用文字模式的話便會發現自己在 root 的目錄下，若是使用 X 視窗的使用者的享受和方便度就比文字視窗要好的多，雖然是在圖形介面下，但是所操作的環境仍在自己的家目錄下，但是一些對於 Linux 的基本的文字模式的操作和觀念可不能免，而且是絕對需要的！

## 6.2 系統殼 (shell)

登入系統之後，您應該可以看到系統殼 (shell) 的提示符號：

```
Red Hat Linux release 6.0  + CLE v0.8
Kernel 2.2.5-15CLE on an i686
login: root
Password:
[root@net53 /root]#
```

這個提示符號最前面是使用者名稱，"@" 後面緊跟著的是所在主機的名稱(hostname)，再來是目前所在的目錄與最後的一個"#"符號，每次你對 UNIX 下達一個新的指令，執行完後便會又再出先一個新的提示符號，以接受下一個新的指令，Un*x 系統的指令其實都是一些可執行檔，這些程式通常放在 /bin/, /usr/bin/ 或 /usr/X11R6/bin/ 底下，當您在提示符號下鍵入一段文字並按下【Enter】以後，系統殼就會去幾個預設的目錄底下找尋同名的程式，然後執行這個程式。

提示符號通常是 # 或 $ (根據不同的 shell 而相異，此舉為 bourne shell，也可依喜好更改)，以 bourne shell 為例， # 符號代表著目前最高使用者的提示符號，一般的使用者的提示符號為 $。

這時相信初學者的疑問是什ㄇ是系統殼(shell)ㄚ？大體來講 Shell 算是一種環境，使用者每次進系統以後，登入程式都會自動幫您呼叫一個系統殼來使用，shell 程式提供使用者一個操作介面跟環境，好像提供一個 "殼" 給您居住一樣，所以叫做 shell(系統殼)，並且提供您一個交談式的操作介面，讓您輸入指令，幫您執行指令，如果講細一點，花一小節來說明也不為過。

### 6.2.1 何謂 Shell

對於 Shell 我們常常用它，但卻對它不是很了解，用翻譯軟體去查它的意義，所得的譯詞為~貝類的外殼，它有保護內部的意義，但它這個字用在 Unix 系統族類作為命令解譯的程式名稱，也就是說 Shell 把我們在 Unix 的系統下所下的指令，轉譯成 Unix 系統核心能夠了解的意義，再交由系統執行，所以就如同一層殼般的保護系統核心與外界的溝通，也可防止使用者因為錯誤的操作而造成系統的傷害。再說 UNIX 將 shell 獨立於核心程式之外，使得它就如同一般的應用程式，可以在不影響作業系統的情況下被修改，更新版本或是添加新的功能，都不用怕傷害到系統。而且 Shell 同時也是一個功能強大的語言，文法有點類似 C 語言，我們可以按照它的語法，自己寫一個 shell 程式，不用編譯器就可以執行。

如果還不太了解的話，以 DOS 為例，它的 shell 就是 command.com，這樣子用過 DOS 的讀者應該心有戚戚焉了吧！

在 UNIX 系統族類下，比較普遍的 shell 有 Broune shell、C shell~~~等等，不勝枚舉，在本章將介紹最常使用的 Broune shell。

## 6.3 Bash

Bash 是 Bourne Again SHell 的縮寫， 而 Bourne Shell (sh) 是個最老牌的 Unix shell， 早年由 Stephen Bourne 寫成， bash 顧名思義，便是 sh 的相容改良品。

Bash 也是個 public domain shell， 由 Free Software Foundation 之 GNU 計畫所衍生出來， 其最終目的是希望能成為符合 IEEE Posix Shell 的成品。目前這個 shell 廣泛在學術校園裡使用， 例如 Linux 的各大 distribution 都有將 bash 收錄。 而且 bash 提供了所有 C shell (csh)、Korn shell (ksh) 的 interactive 特色，而且語法上與 Bourne shell (sh) 完全相容。您應該可以在 Linux 上看到 /bin/bash， 而且可能發現 /bin/sh 也是個 link 檔。RedHat 預設使用的 shell 也是 bash，在開始的時候提供了使用者一個很好用的操作環境。

那冂 shell 的啟動過程為何乁？了解這個過程的話,想要改變自己的使用環境的使用者可就方便了！其實在我們一開始登入系統的時候,登入的程式就會建立起一個初始化的環境，這個環境是由定義工作環境的環境變數傳到 shell 所構成的。

那冂所謂的環境變數是什冂乁？其實 shell 本身也有一組變數用來記錄她所需的儲存的資訊，如果所紀錄的資料是關於 shell 執行環境的各種資訊，這些變數我們就稱之為環境變數(environment variables)，因此我們所登入的使用者名稱、家目錄、**登入所使用的 shell 程式**，這些都當成環境變數傳給 shell 去執行建構出一個工作環境，但這執行的 shell 跟**登入所使用的 shell 程式**是不一樣的。

等登入的 shell 啟動後，再讀取/etc/下的 shell 起始環境設定檔，接著檢查你的登入目錄(家目錄),看看是否有任何關於 shell 的起始設定檔案存在,如果有的話，便會被執行。這類的起始設定檔被用來進一步的個別化使用者的環境，等到上述的檔案都被執行過後，提示符號才會出現在螢幕上。這時讀者可能又有一個疑問，為什冂要個別化使用者的環境乁？因為 Linux 是多人多工的，假設每個使用者都指定用 bsah 當作登入的 shell，那冂每個人除了使用者名稱、家目錄不同外，還有什冂不一樣的乁？為了能夠打造量身訂做的使用環境，這些起始設定檔便佔著重要的角色！

我們現在以 bash 當作登入的 shell 為例，說明登入讀取執行的起始設定檔的順序，$HOME 這在代表著家目錄的變數，也就是你目前的登入目錄！

**啟動時：**

/etc/profile - login shell 才會讀取

$HOME/.bash_profile - login shell 才會讀取

$HOME/.bash_login - 如果沒有 .bash_profile，則會在 login 時讀取

$HOME/.profile - 如果沒有 .bash_profile，則會在 login 時讀取

在這些起始設定檔中，/etc 下的 profile 是系統管理者為大家設定的，一般使用者無法做任何更動，如果對系統管理者的設定覺得不夠或不符需要，則可在個

人帳號的.profile、.cshrc、.login 增加或修正，尤其是對 csh 及 tcsh 的使用者而言，因為在/etc 下並沒有 start files，所以也不會有系統管理者設定好的環境可以使用，使用者就必須自行設定所需要的所有的變數與指令，否則常會有寸步難行的感覺，最常見的如許多指令都因找不到而不能用(因為沒有設 path)、<backspace>鍵無法使用 (因<backspace>鍵的作用沒有定義)...等。

除了/etc/profile 這個檔，其餘檔案都已經在你的登入目錄(家目錄)中，這些檔案檔名前面都有一個 ．，這個點代表此檔為隱藏檔，所以一般而言你直接用 ls 的指令而不加選項去查詢的話，是看不到的，你必須執行 **ls –la** 才看得到。

然而在我們登出的時候也會讀取一個檔案並且執行，以 bash 為例此檔為 $HOME/.bash_logout，相較於剛才介紹的登入讀取檔案，它便顯得無關緊要，我們甚至可以忽略它，這個檔案的用途通常用來放一些格言或有趣的句子，如此可以讓我們登出的時候不會顯得太無聊！

**結束之前：**

$HOME./ .bash_logout - login shell 讀取之

**其他：**

$HOME/ .inputrc - Readline 初始化之時

再來我們對一些起始設定檔做說明。

/etc/profile

example$ cat /etc/profile

```
#     /etc/profile 檔之範例說明
#
#     僅敘述一些重要的設定，非完整設定，
#     故使用上如有任何問題，筆者完全不負責任

PATH="$PATH:/usr/X11R6/bin:." # 在最後添加 . 表示「允許執行現行目錄下的程式」，安裝某
                                                    些程式時會較方便，不過系統安全問題請自負囉！
stty pass8
export LANG="C"
export LC_CTYPE="iso-8859-1" # 這裡是讓您日後可以於命令列輸入中文
export TMOUT="1200"        # 設定 auto-logout 的時間，單位是秒
eval `dircolors -b`
alias ls='/bin/ls -F --color'      # 有彩色 ls 的功能囉
```

對於上述的參數，筆者對於最常用的做一些部分的說明：

**1. PATH**

設定執行檔的尋找路徑，若現行目錄也要包含在尋找的路徑中，則需在設定路徑時給予一'.'，表示現行目錄亦包含在尋找的路徑中，如：

PATH=.:/bin:/usr/local/bin:/usr/ucb

或

set path=(. /bin /usr/local/bin /usr/ucb)

**2. EDITOR**

設定想要使用的編輯器，如

EDITOR=/usr/ucb/vi

或

setenv EDITOR /usr/ucb/vi

**3. HOME**

設定個人帳號 home directory 的所在位置，通常在 login 時即參考/etc/passwd 而給予一設定值，因此除非使用者有某些特殊需求，通常不會重新設定。

**4. SHELL**

同樣是在 login 時即會參考/etc/passwd 的內容而自動設定，使用者即使重新設定這個變數的內容，對帳號的使用也不會有任何影響。

**5. 提示符號的設定**

在 sh 中： PS1="[STRING]"

PS2="[STRING]"

在 csh 中： set prompt="[STRING]"

當然以上所列的只是使用得較為廣泛的變數，若您覺得這些不夠用，可用 man sh、man csh、man tcsh、man bash 指令查閱各 shell 的說明，在其中您將可看到各個 shell 所包含的變數及各變數的用法。

若不清楚某一變數的設定內容，可用'echo $[VAR]'來觀察變數的內容。

例如：

```
echo $PATH
```

如果您是使用 csh / tcsh， 那麼 /etc 目錄下同樣有類似的設定檔，像是 csh.cshrc、 csh.login 檔案， 有興趣的朋友可以如法泡製。

上述說明的只是 bash 的「系統設定檔」， 一般情況下，使用者還是可以在 $HOME 目錄下， 自行設定諸如 .profile 或 .bashrc 之類的檔案， 以便打造量身訂做的使用環境。

在/etc/shells 這個檔案說明了系統提供哪些 shell 讓使用者使用，系統管理員須檢查本檔案是否列明所有合法的 shell 程式。

然而我們在提示符號下要如何體驗 bash 的好處和便利ㄋ？清楚 bash 的功能是事半功倍的開始喔！

**1. 命令列編輯程式：**

bash 的命令列編輯能力是內建的，這種功能以現在來說，可說是理所當然，就是你在提示符號下，可對未執行的指令字元任意的修改，即使拼錯了也不需要重新輸入整個命令，只需在執行指令前使用左右方向鍵移動游標，用【backspace】或【del】刪除游標前一個字元來編輯打錯的指令編輯功能糾正錯誤即可，這尤其適合於冗長的路徑名稱當作參數的命令時。

此外，【ctrl】+【a】 會把游標移到指令的最前面，用【ctrl】+【e】可以把游標移動到指令的最後面，非常方便。用【ctrl】+【c】可以中斷目前在編輯的這行指令，跳到一行新的提示符號重新輸入，此外，在程式執行中，【ctrl】+【c】還可以用來強行中斷程式的執行，回到提示符號。對於這種複合鍵，筆者順便在此做一些說明：

【ctrl】+【C】：停止命令或程式的執行。

【ctrl】+【D】：停止輸入或簽出系統也就是所謂的 logout！

【ctrl】+【S】：暫時停止螢幕的輸出，始螢幕內容滯留，不向上捲動。

【ctrl】+【Q】：恢復螢幕輸出。

【ctrl】+【\】：終止一個處理程序的執行。

當你按這些複合鍵時，有時螢幕會顯示 ^ 這樣的符號，比如說【ctrl】+【c】啦，就會出現 ^c 的字樣，也就是說 ^ 代表著【ctrl】鍵。

**2. 命令歷程(command history)：**

所謂的命令歷程就是把你曾經輸入過的指令紀錄起來，方便日後的查詢與使用。而這也是我們最常用的功能，就好像是 DOS 模式中的 Doskey 功能一樣，但是功能更強喔！要是我們懶的打相同或類似的指令，只要按方向鍵中的向上鍵就

可以叫出前一個指令，再按一次向上鍵就可以叫出更前一個指令，依此類推，用向下鍵可以回到下個指令，所以用上、下鍵就可以選擇以前輸入過的指令，再配合編輯指令的功能就可以少打很多字，這對於懶得動手的人可是一大福音ㄋ！

況且它以 history 的工具程式記錄了你所執行過的指令，History 程式是一種短期記憶，記錄了以最近執行的指令，也不完全是指令而已，只要你曾在提示符號下鍵入字元並且按【Enter】鍵執行過的話，都會被紀錄下來喔！

使用方法是在提示符號下我們鍵入 **history** 指令後，螢幕便會列出一段或一大串的命令的歷程(看你打過多少囉！)，如下：

```
[root@net53 down]# history
    1  cd /usr
    2  ls
    3  mkdir home
    4  cd home
    5  ls
    6  cd /usr/home
    7  mkdir a8630257
    8  chown a8630257 a8630257
    9  ls
   10  ls –la
```

命令歷程是由 1 開始編號，預設值應該是 500，編號的號碼越大，表示所執行過的指令離目前時間最近，使用者可以這樣來辨別。

**3. 自動補全功能：**

自動補全的功能,可說是造福所有 linux 的使用者,記得筆者剛開始接觸 linux 有時候,曾為這個功能感動的痛哭流涕ㄋ！感動的原因是假設我們要下的指令很長，或者指令後面要給的檔名很長，這個時候只要按一個【Tab】鍵，bash 就會在可能的指令或檔名裡面找尋匹配的，找到的話就會自動幫您補齊，當然啦！使用者打的字元越多，bash 收循匹配的檔名或指令就越容易找到。

比如說有一個檔名或指令叫做 Iorikyo，使用者只記得 iori 這四個字元的話，在提示符號下輸入 iori 之後在按【Tab】鍵，如果 Iorikyo 這個名字是獨一無二的話，這時就會幫你補齊 kyo 這三個字囉！要是不是獨一無二的話，比如說還有 Iorikyo1、Iorikyo2~~~等，這時你按【Tab】鍵當然不會跑出來囉！這時你必須連續按兩下【Tab】，你會聽到嗶一聲 bash 就會把所有符合的名稱給列出來，然後你在參考 bash 所列出來的名稱再輸入更多字元以便 bash 來辨認你所要的名稱！

**4. 特殊字元：**

在 UNIX 系統裡，有一組更完備的特殊意義字元，這就是所謂的萬用字元(meta – character)，曾經用過 DOS 系統的人，應該感到不陌生，像 * 這種符號可以方便我們執行系統檔案的查詢，所以囉！~在此筆者將介紹有關於 UNIX 的

特殊字元，如果善加利用這些特殊字元的話，跟 linux 溝通可說是得心應手！使用萬用字元的好處為：

- 減少輸入【key　in】時間
- 鼓勵使用者使用良好的檔案命名方式
- 簡化 Shell Script 的設計工作

現在我們要列出一些常用的萬用字元的意義，供您參考：

| 萬用字元 | 意　　義 |
|---|---|
| .. | 上一層目錄，與 cd 指令配合用比較多 |
| . | 表目前工作的目錄 |
| * | 任意長度的字元 |
| ? | 長度為一個的任意字元 |
| [ .. ] | 括號內的一個字元 |
| \m | 等於某個萬用字元，如 *、? 等 |
| [ a- z ]* | 小寫字母開頭的所有字串 |
| \ | 跳脫符號，用以解除特殊字元的特殊意義 |
| ~ | 使用者目錄 |
| ; | 分隔符號，當命令列有多個指令時，做分隔用 |
| $ | Bourne shell 的提示符號，同時也為 shell 語言的位址參數 |
| # | 做註解用 |
| \| | 建立一個管線，使一命令的輸出作為另一個命令的輸入 |
| & | 將命令以幕後的方式執行 |
| > | 將命令的輸出重導入檔案中 |
| < | 將命令的輸入流指定為由檔案中載入，跟>相反就對了！ |
| >> | 將命令的輸出家在一個已經存在的檔案後面 |
| { .. } | 括號內的一個字串 |

接下來我們就舉幾個實例來幫您了解這些萬用字元並對其中幾個做詳加說明，其他的在下一章令介紹做應用介紹：

- ls　-l　*.c

列出所有以“.c”結尾的檔案或目錄，但是“*”不能用於以“.”開頭的檔案與目錄，像“.hello.c”的檔案就不包含在“*.c”內。

- ls -l /dev/tty?

“？”是長度為一的任意字元，所以像 tty1、tty2、ttya、 ttyx 等都是，所以這行指令說要列出所有以 tty 為開頭，且其 後只有一個字元的檔案。

- ls -l [A-D]*

此指令匹配方式可分為兩部份，第一是：“[ A-D] ”是指 A、B、C、D 其中之一；第二是：“ * ”是指任意字串。所以整個意思就是以 A、B、C、D 為開頭的任意字串。

- ls -l [Yy][Ee][Ss]

此指令就等於：ls -l YES YeS Yes Yes yES yeS yEs yes。

- ls -l /dev/{tty,fd,hda}0

此指令就等於：ls -l /dev/tty0 /dev/fd0 /dev/hda0。{ }與[ ]類似，不過{}是為括號內
的某一個『字串』，而[ ]是為括號內的某一個『字元』。

- more ~/readme

若使用者目錄【home directory】是為/root，則此指令就等於：more /root/readme。
此外，'~' 後面加上使用者名稱的話，可以用來指到系統上其他使用者的家目錄，"~john/" 就相當於 "/home/john"。

- more *readme

嚴格而言，倒斜線“\”是一種引號，若加在一個萬用字元之前，則萬用字元就變成一般字元。像此例中，“ * ”若視為一個萬用字元，就是代表任意字串，而今它之前有“\”於是就變成一般字元即是一個星號，不再是任意字串。

【 ; 】在 shell 是扮演將命令與命令分隔的角色，利用它使用者可以在命令列下輸入數個命令，而不用再換行輸入，例如：

ls ; date ; df

在同一行輸入這三種指令後，你可以看到先列出目錄列表，執行完後再出現日期，執行完又出現硬碟的使用情形！

【 # 】是用來做註解用的，在使用 sheel 所提供的命令語言( command language )寫批次檔 (batch file) 時，以【 # 】字原來指定程式的說明，接在【#】號之後的一切敘述都僅供說明使用，將不被程式所執行，日後讀者將會看到一些設定檔案，檔案裡面的說明皆是用此種方式表示之！

【 | 】 字元 (管線) 唸做 pipe，他是 unix 系統為使兩個單獨的命令共用的資料，或是讓數個命令彼此協同完成一件工作所提供的特殊工具符號。它使某一個命令的輸出成為另一命令的輸入，換言之他擔任兩個命令間通訊的介面。此符號

在指令簡介的 ls 指令中會以實例介紹之，讀者到時候即可體會筆者欲說明的意義！

【 & 】這個符號的作用是將命令以幕後的方式執行。所以當你下達指令的時候，系統不會像往常一立刻執行你的命令，而是會傳回你的一個正整數 (job ID 工作識別碼)，然後再回提示符號下等在你的下一個指令。當幕後的命令取得了所有必要的資源後便會進入準備執行的狀態，只要 CPU 一有空就立刻執行，因為它何時被執行且執行結果都不易掌握，有如隔了一層帷幕般，所以叫幕後，等到執行完成後，系統會在螢幕上顯示 done 的字樣，並印出該工作的處理程序識別碼。但是在幕前執行的話，當你命令下達後，unix 會立刻執行你的命令，在該命令完成前，你只能乾做在螢幕前什冂事也不能做，除非你再換另一個虛擬終端來工作，或者按【ctrl】+【C】強制停止命令或程式的執行，才能執行下一個命令！

當你進行一項很耗時且不需要監督的工作時，例如用 ftp 傳檔啦！將工作放到幕後執行是一件不錯的方式，如果你有簽離系統的必要的話，你可以配合 nohup 的指令使你在簽離後該幕後命令能繼續運作不被打斷，這指令的用法下一章中說明！

在對【 < 】和【 > 】這兩個重導符號說明之前，我們務必了解一些觀念，unix 在設計理念上的一個創舉就是將輸出與輸入系統放在檔案系統下，作為檔案系統的一個子系統(sub system)，因此使用 shell 所提供的重導符號，可以使檔案取代鍵盤的輸入成為輸入資料的來源，或使檔案代替終端機成為資料輸出的對象。

【 < 】符號是將輸入的資料由預設的裝置重導為由檔案讀入，也就是說可以用來將一個檔案的內容當作一個程式的輸入，讓程式從一個檔案裡面去讀取本來該用手動輸入的資料。比方說：wc < text。而【 > 】符號則擔任相反的功能。

除了重導符號，unix 也提供了【 >> 】這個特殊字元，它能將兩個檔案合併在一起。比如說你目錄中有 F1 和 F2 這兩個檔案，那冂命令：cat F1>>F2，將使 F1 的內容緊跟著 F2 檔案內容之後，假如指定的檔案 F2 不存在的話，會新建一個 F2，兩者差異在於前者會破壞原先的檔案，把原有的檔案內容蓋過去，而後者會把輸出結果附在檔案後面，不會破壞原先的資料。

## 6.4 shell 的程式設計簡介

對於一些玩 Linux 的人來說，不會使用 shell 來設計一些程式，就不用玩 Linux 了，因為 shell 強大的語言，使得每個使用者都可以在上面設計自己想要的程式，利用 shell 所發展的小工具程式，來完成原本需要大量軟體開發的工作，這一點特色，使得 Unix 族類才會那冂受大家歡迎，所以不要以為 shell 祇是跟核心溝通的程式而已，它同時也是一個功能強大的語言，有時候為了方便私人的作業，便可隨意設計一支 Shell Script，Shell Script 算得上是最基本、最強大、運用最廣泛

的一個。它運用範圍之廣，不但從系統啟動、程式編譯、定期作業、上網連線，甚至安裝整個 Linux 系統，都可以用它來完成。若是你學有所成的話，你也可以把之前所說的 bash 起始設定檔改得有你個人的專屬使用環境！

因為 Shell Script 是利用您平日在使用的一些指令，將之組合起來，成為一個"程式"。如果您平日某些序列的指令下得特別頻繁，便可以將這些指令組合起來，成為另一個新的指令。這樣，不但可以簡化並加速操作速度，甚至還可以乾脆自動定期執行，大大簡化系統管理工作。

雖然說 shell 的程式很多種，但是每一個都大同小異，只要學會一種，其他的就很容易上手了，如今最多人也是 Red Hat Linux 預設的 shell 是 bash，所以現在我們要專門對此 bash 做教學，在教學的過程中，會常常用到第四章的指令，所以若遇到不懂的請翻到第四章，這樣下來很快的你會發現，你已經把第四章的主要指令都學起來了，可說是一舉兩得。

### 6.4.1 Shell Script 的"Hello World"的撰寫

"Hello world" Shell Script
照傳統程式教學範例，這一節介紹 Shell Script 的"Hello World"如何撰寫。

---

```
#!/bin/sh
# Filename : hello
echo "Hello world!"
```

---

大家應該會注意到第一行的"#!/bin/sh"。在 UNIX 下，所有的可執行 Script，不管是那一種語言，其開頭都是"#!"，例如 Perl 是"#!/usr/bin/perl"，tcl/tk 是"#!/usr/bin/wish"，看您要執行的 Script 程式位置在那裡。您也可以用"#!/bin/bash"、"#!/bin/tcsh"等等，來指定使用特定的 Shell。
echo 是個 bash 的內建指令。

---

接下來，執行 hello 這個 script:
要執行一個 Script 的方式有很多種。

---

**第一種：將 hello 這個檔案的權限設定為可執行。**

```
[root@net53 /root]# chmod 755 hello
```

執行

```
[root@net53 /root]# ./hello
hello world
```

---

**第二種：使用 bash 內建指令"source"或"."。**

[root@net53 /root]# source hello
hello world
或
[root@net53 /root]# . hello
hello world
---
**第三種：直接使用 sh/bash/tcsh 指令來執行。**
[root@net53 /root]# sh hello
hello world
或
[root@net53 /root]# bash hello
hello world
---
**Bash 執行選項：**
-c string : 讀取 string 來當命令。
-i : 互動介面。
-s : 由 stdin 讀取命令。
- : 取消往後選項的讀取。
-norc : 不要讀~/.bashrc 來執行。
-noprofile : 不要讀/etc/profile、~/.bash_profile、~/.bash_login、~/.profile 等等來執行。
-rcfile filename : 執行 filename，而非~/.bashrc
-version : 顯示版本。
-quiet : 啟動時不要囉哩囉唆。
-login : 確保 bash 是個 login shell。
-nobraceexpansion : 不要用 curly brace expansion({}符號展開)。
-nolineediting : 不用 readline 來讀取命令列。
-posix : 改採 Posix 1003.2 標準。

### 6.4.2 shell scripe 的應用實例

現在我們來講一個用於自動備份的 Shell Script，利用 Shell Script 搭配 crond 指令來作定期備份的工作。要作定期性的工作，在 UNIX 上，就是一定要與 crond 指令的搭配運用。再開始之前，首先我們先來研究如何對系統進行備份。

要對系統進行備份，不外乎便是利用一些壓縮工具。在許多 UNIX 系統上，tar 及 gzip 是 de facto 的資料交換標準。我們經常可以看見一些 tar.gz 或 tgz 檔，這些檔案，被稱為 tarball。當然了，您也可以用 bzip2、zip 等等壓縮工具來進行壓縮，不必限定於 gzip。但 tar 指令配合 gzip 指令是最普遍的，也是最方便的方

式。

要將我們想要的資料壓縮起來，進行備份，可以結合 tar 及 gzip 一起進行。方式有很多種，最常用的指令是以下這一種：

tar -c file/dir ... | gzip -9 > xxxx.tar.gz

您也可以分開來做:

tar -r file/dir ... -f xxxx.tar

gzip -9 xxxx.tar

或

tar -r file/dir ... -f xxxx.tar

gzip -9 < xxxx.tar > xxxx.tar.gz

---

在瞭解過 Linux 下檔案備份的基本知識後，我們來寫一個將檔案備份的 Script。

```
#!/bin/sh
# Filename : backup
DIRS="/etc /var /your_directories_or_files"
BACKUP="/tmp/backup.tgz"
tar -c $DIRS | gzip -9 > $BACKUP
```

其中 DIRS 放的是您要備份的檔案及目錄，BACKUP 是您的備份檔。可不要將/tmp 放進 DIRS 中，那樣做，您是在做備份的備份，可能將您的硬碟塞爆。

---

接下來測試

```
[root@net53 /root]# chmod 755 backup
[root@net53 /root]# . /backup
```

執行完成後在/tmp 就會有一個 backup.tgz，裡面儲存了您重要的資料。您可用

gzip -dc /tmp/backup.tgz | tar -vt

或

tar vtfz /tmp/backup.tgz

來看看裡面的檔案列表。

要解開時，可用以下指令來完成復原:

gzip -dc /tmp/backup.tgz | tar -xv

或

tar xvfz /tmp/backup.tgz

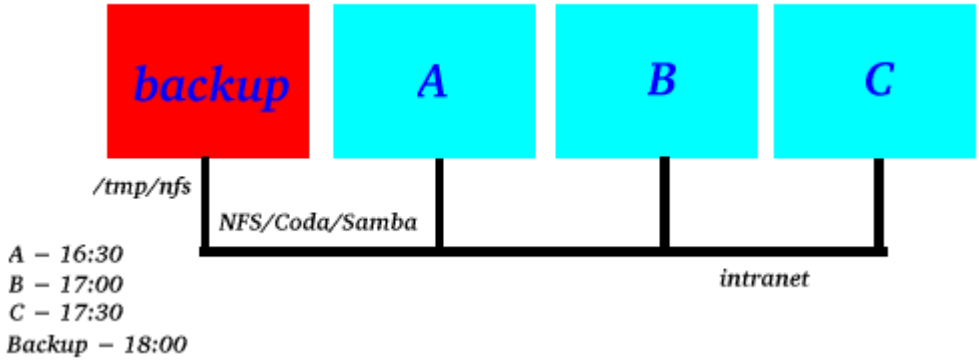
備份通常是僅備份系統通常最重要的部份，/etc 可說是不可缺少的一部份。另外，看您系統中有那些重要的資料需要備份。通常來說，您沒有必要備份/bin、/sbin、/usr/bin、/usr/sbin、/usr/X11R6/bin 等等這些執行檔目錄。只要備份您重要的檔案即可，別把整個硬碟備份，那是蠻呆的動作。

---

如果您有許多台機器，可利用其中一台任務較輕的內部網路主機，做為主要

備份主機。將所有機器都自動執行備份，然後利用 NFS/Coda/Samba 等網路檔案系統，將備份的資料放到該備份機器中，該機器則定時收取備份資料，然後您再由該機器中進行一次備份。

這裡是整個系統備份方案的圖示。




在您進行之前，先瞭解一下，系統中那些是要備份的，那些是不需要的。

---

**新的 backup**

```
#!/bin/sh
HOSTNAME=`hostname`
DIRS="/etc /var /your_important_directory"
BACKUP="/tmp/$HOSTNAME.tgz"
NFS="/mnt/nfs"
tar -c $DIRS | gzip -9 > $BACKUP
mv -f $BACKUP $NFS
```

---

備份主機內的 Script : collect_backup

```
#!/bin/sh
NFS="/mnt/nfs"
BACKUP="/backup"
mv -f $NFS/*.tgz $BACKUP
```

在此，您不能夠將所有備份都直接放在/mnt/nfs，這是危險的。萬一任一台機器不小心將/mnt/nfs 里所有內容刪除，那麼備份就會消失。因此，您需要將/mnt/nfs 移到一個只有該備份主機可存取的目錄中。

當這些個別的 Script 都測試好以後，接下來我們將他們放到 crontab 裡面。找到您的 crontab，它的位置可能在/var/spool/cron/crontabs/root、/etc/crontab、/var/cron/tabs/root。

在 crontab 中選擇以下之一加入(看您定期的時間):

Slackware : /var/spool/cron/crontabs/root

```
01 * * * *   /full_backup_script_path/backup 1> /dev/null 2> /dev/null # 每小時(太過火一點)
```

30 16 * * *　/full_backup_script_path/backup 1> /dev/null 2> /dev/null # 每日 16:30，下班前備份
30 16 * * 0　/full_backup_script_path/backup 1> /dev/null 2> /dev/null # 每周一 16:30
0 5 1 * *　/full_backup_script_path/backup 1> /dev/null 2> /dev/null # 每月一號 5:0
RedHat/Debian : /etc/crontab
RedHat 可直接將 backup 放入/etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly。或採用如上加入/etc/crontab 的方式:
有關 crontab 的用法，可查"man 5 crontab"，在此不詳述。
備份主機的設定類同。
注意：所有機器不要同時進行備份，否則網路會大塞車。備份主機收取備份的時間要設為最後，否則會收不到備份資料。您可以在實作後，將時間間隔調整一下。

看看，兩個小小不到三行的 Shell Script，配合 cron 這個定時工具。可以讓原本需要耗時多個小時的人工備份工作，簡化到不到十分鐘。善用您的想像力，多加一點變化，可你讓您的生活變得輕鬆異常，快樂悠哉。

### 6.4.3 檔案系統檢查

系統安全一向是大多數電腦用戶關心的事，在 UNIX 系統中，最重視的事，即系統中有沒有"木馬"(Trojan horse)。不管 Trojan horse 如何放進來的，有一點始終會不變，即被放置木馬的檔案，其檔案日期一定會被改變，甚至會有其它的狀態改變。此外，許多狀況下，系統會多出一些不知名的檔案。因此，平日檢查整個檔案系統的狀態是否有被改變，將所有狀態有改變的檔案，以及目前有那些程式正在執行，自動報告給系統管理員，是個避免坐上"木馬"的良方。

---

```
#!/bin/sh
# Filename : whatever_you_name_it
DIRS="/etc /home /bin /sbin /usr/bin /usr/sbin /usr/local /var /your_directory"
ADMIN="email@your.domain.com"
FROM="admin@your.domain.com"
# 寫入 Sendmail 的標頭
echo "Subject: $HOSTNAME filesystem check" > /tmp/today.mail
echo "From: $FROM" >> /tmp/today.mail
echo "To: $ADMIN" >> /tmp/today.mail
echo "This is filesystem report comes from $HOSTNAME" >> /tmp/today.mail
# 報告目前正在執行的程式
ps axf >> /tmp/today.mail
# 檔案系統檢查
```

echo “File System Check” >> /tmp/today.mail
ls –alR $DIRS | gzip –9 > /tmp/today.gz
zdiff /tmp/today.gz /tmp/yesterday.gz >> /tmp/today.mail
mv –f /tmp/today.gz /tmp/yesterday.gz
# 寄出信件
sendmail –t < /tmp/today.mail
然後把它放到一個不顯眼的地方去，讓別人找不到。
把它加入 crontab 中。
30 7 * * *  /full_check_script_path/whatever_you_name_it 1> /dev/null 2> /dev/null
#上班前檢查
有些檔案是固定會更動的，像/var/log/messages、/var/log/syslog、/dev/ttyX 等等，不要太大驚小怪。

### 6.4.4 高階的 shell 寫法

示範了幾個簡單的 Shell Script，相信您應該對 Shell Script 有點概念了。現在我們開始來仔細研究一些較高等的 Shell Script 寫作。有一些符號，例如"$"、">"、"<"、">>"、"1>"、"2>"符號的使用，之前有介紹過，讀者可以再複習一下。

- **3.4.4.1 控制迴圈 - for**

---

**for name [ in word; ] do list ; done**
控制迴圈。
word 是一序列的字，for 會將 word 中的個別字展開，然後設定到 name 上面。list 是一序列的工作。如果[in word;]省略掉，那麼 name 將會被設定為 Script 後面所加的參數。

---

**範例一：**

```
#!/bin/sh
for i in a b c d e f ; do
    echo $i
done
```

它將會顯示出 a 到 f。

---

**範例二：另一種用法，A-Z**

```
#!/bin/sh
WORD="a b c d e f g h i j l m n o p q r s t u v w x y z"
```

```
for i in $WORD ; do
   echo $i
done
```

這個 Script 將會顯示 a 到 z。

---

**範例三：修改副檔名**

如果您有許多的.txt 檔想要改名成.doc 檔，您不需要一個一個來。

```
#!/bin/sh
FILES=`ls /txt/*.txt`
for txt in $FILES ; do
   doc=`echo $txt | sed "s/.txt/.doc/"`
   mv $txt $doc
done
```

這樣可以將*.txt 檔修改成*.doc 檔。

---

**範例四：meow**

```
#!/bin/sh
# Filename : meow
for i ; do
      cat $i
done
```

當您輸入"meow file1 file2 ..."時，其作用就跟"cat file1 file2 ..."一樣。

---

**範例五： listbin**

```
#!/bin/sh
# Filename : listbin
for i in /bin/* ; do
      echo $i
done
```

當您輸入"listbin"時，其作用就跟"ls /bin/*"一樣。

---

**範例六：/etc/rc.d/rc**

拿一個實際的範例來說，Red Hat 的/etc/rc.d/rc 的啟動程式中的一個片斷。

```
for i in /etc/rc.d/rc$runlevel.d/S*; do
      # Check if the script is there.
      [ ! -f $i ] && continue
      # Check if the subsystem is already up.
```

```
    subsys=${i#/etc/rc.d/rc$runlevel.d/S??}
    [ -f /var/lock/subsys/$subsys ] || \
    [ -f /var/lock/subsys/${subsys}.init ] && continue
    # Bring the subsystem up.
      $i start
done
```

這個範例中，它找出/etc/rc.d/rcX.d/S*所有檔案，檢查它是否存在，然後一一執行。

#### 6.4.4.2 流程控制 – case

case word in [ pattern [ | pattern ] ... ) list ;; ] ... esac

case/esac 的標準用法大致如下:

```
case $arg in
    pattern | sample) # arg in pattern or sample
    ;;
    pattern1) # arg in pattern1
    ;;
    *) #default
    ;;
esac
```

arg 是您所引入的參數,如果 arg 內容符合 pattern 項目的話,那麼便會執行 pattern 以下的程式碼，而該段程式碼則以兩個分號";;"做結尾。

可以注意到"case"及"esac"是對稱的,如果記不起來的話,把"case"顛倒過來即可。

---

**範例一：paranoia**

```
#!/bin/sh
case $1 in
        start | begin)
           echo "start something"
        ;;
        stop | end)
           echo "stop something"
        ;;
        *)
           echo "Ignorant"
        ;;
esac
```

執行

```
[root@net53 /root]# chmod 755 paranoia
[root@net53 /root]# ./paranoia
Ignorant
[root@net53 /root]# ./paranoia start
start something
[root@net53 /root]# ./paranoia begin
start something
[root@net53 /root]# ./paranoia stop
stop something
[root@net53 /root]# ./paranoia end
stop something
```

請注意 / 之前有個 . 喔！

---

**範例二：inetpanel**

許多的 daemon 都會附上一個管理用的 Shell Script，像 BIND 就附上 ndc，Apache 就附上 apachectl。這些管理程式都是用 shell script 來寫的，以下示範一個管理 inetd 的 shell script。

```
#!/bin/sh
case $1 in
    start | begin | commence)
        /usr/sbin/inetd
    ;;
    stop | end | destroy)
        killall inetd
    ;;
    restart | again)
        killall -HUP inetd
    ;;
    *)
        echo "usage: inetpanel [start | begin | commence | stop | end | destory | restart |
again]"
    ;;
esac
```

---

**範例三：判斷系統**

有時候，您所寫的 Script 可能會跨越好幾種平台，如 Linux、FreeBSD、Solaris

等等，而各平台之間，多多少少都有不同之處，有時候需要判斷目前正在那一種平台上執行。此時，我們可以利用 uname 來找出系統資訊。

```
#!/bin/sh
SYSTEM=`uname -s`
case $SYSTEM in
      Linux)
            echo "My system is Linux"
            echo "Do Linux stuff here..."
      ;;
      FreeBSD)
            echo "My system is FreeBSD"
            echo "Do FreeBSD stuff here..."
      ;;
      *)
            echo "Unknown system : $SYSTEM"
            echo "I don't what to do..."
      ;;
esac
```

#### 6.4.4.3 流程控制 – select

select name [ in word; ] do list ; done

select 顧名思義就是在 word 中選擇一項。與 for 相同，如果[in word;]省略，將會使用 Script 後面所加的參數。

---

**範例：**

```
#!/bin/sh
WORD="a b c"

select i in $WORD ; do
   case $i in
      a)
         echo "I am A"
      ;;
      b)
         echo "I am B"
      ;;
      c)
```

```
        echo "I am C"
      ;;
      *)
        break;
      ;;
   esac
done
```

執行結果

```
[root@net53 /root]# ./select_demo
1) a
2) b
3) c
#? 1
I am A
1) a
2) b
3) c
#? 2
I am B
1) a
2) b
3) c
#? 3
I am C
1) a
2) b
3) c
#? 4
[root@net53 /root]#
```

#### 6.4.4.4 返回狀態 - Exit Status

在繼續下去之前，我們必須要切入另一個話題，即返回狀態值 - Exit Status 。因為 if/while/until 都遷涉到了使用 Exit Status 來控制程式流程的問題。

許多人都知道，在許多語言中(C/C++/Perl....)，都有一個 exit 的函數，甚至連 Bash 自己都有個 exit 的內建命令。而 exit 後面所帶的數字，便是返回狀態值 - Exit Status。

返回狀態值可以使得程式與程式之間，利用 Shell script 來結合的可能性大增，利

用小程式，透過 Shell script，來完成很複雜的工作。在 shell 中，返回值為零表示成功(True)，非零值為失敗(False)。

---

舉例來說，以下這個兩個小程式 yes/no 分別會返回 0/1(成功/失敗):

```
/* yes.c */
void main(void) { exit(0); }
/* no.c */
void main(void) { exit(1); }
```

那麼以下這個"YES"的 shell script 便會顯示"YES"。

```
#!/bin/sh
# YES
if yes ; then
      echo "YES"
fi
```

而"NO"不會顯示任何東西。

```
#!/bin/sh
# NO
if no ; then
      echo "YES"
fi
```

---

test express

[ express ]

在 Shell script 中，test express/[ express ]這個語法被大量地使用，它是個非常實用的指令。由於它的返回值即 Exit Status，經常被運用在 if/while/until 的場合中。而在後面，我們也會大量運用到，在進入介紹 if/while/until 之前，有必要先瞭解一下。

其返回值為 0(True)或 1(False)，要看表述(express)的結果為何。

**express 格式**

-b file : 當檔案存在並且屬性是 Block special(通常是/dev/xxx)時，返回 True。
-c file : 當檔案存在並且屬性是 character special(通常是/dev/xxx)時，返回 True。
-d file : 當檔案存在並且屬性是目錄時，返回 True。
-e file : 當檔案存在時，返回 True。
-f file : 當檔案存在並且是正常檔案時，返回 True。
-g file : 當檔案存在並且是 set-group-id 時，返回 True。
-k file : 當檔案存在並且有"sticky" bit 被設定時，返回 True。
-L file : 當檔案存在並且是 symbolic link 時，返回 True。
-p file : 當檔案存在並且是 name pipe 時，返回 True。
-r file : 當檔案存在並且可讀取時，返回 True。
-s file : 當檔案存在並且檔案大小大於零時，返回 True。
-S file : 當檔案存在並且是 socket 時，返回 True。
-t fd : 當 fd 被開啟為 terminal 時，返回 True。
-u file : 當檔案存在並且 set-user-id bit 被設定時，返回 True。
-w file : 當檔案存在並且可寫入時，返回 True。

-x file : 當檔案存在並且可執行時，返回 True。
-O file : 當檔案存在並且是被執行的 user id 所擁有時，返回 True。
-G file : 當檔案存在並且是被執行的 group id 所擁有時，返回 True。
file1 -nt file2 : 當 file1 比 file2 新時(根據修改時間)，返回 True。
file1 -ot file2 : 當 file1 比 file2 舊時(根據修改時間)，返回 True。
file1 -ef file2 : 當 file1 與 file2 有相同的 device 及 inode number 時，返回 True。
-z string : 當 string 的長度為零時，返回 True。
-n string : 當 string 的長度不為零時，返回 True。
string1 = string2 : string1 與 string2 相等時，返回 True。
string1 != string2 : string1 與 string2 不相等時，返回 True。
! express : express 為 False 時，返回 True。
expr1 -a expr2 : expr1 及 expr2 為 True。
expr1 -o expr2 : expr1 或 expr2 其中之一為 True。
arg1 OP arg2 : OP 是-eq[equal]、-ne[not-equal]、-lt[less-than]、
-le[less-than-or-equal]、-gt[greater-than]、-ge[greater-than-or-equal]的其中之一。

---

在 Bash 中，當錯誤發生在致命信號時，bash 會返回 128+signal number 做為返回值。如果找不到命令，將會返回 127。如果命令找到了，但該命令是不可執行的，將返回 126。除此以外，Bash 本身會返回最後一個指令的返回值。若是執行中發生錯誤，將會返回一個非零的值。
Fatal Signal : 128 + signo
Can't not find command : 127
Can't not execute : 126
Shell script successfully executed : return the last command exit status
Fatal during execution : return non-zero

#### 6.4.4.5 流程控制 – if

if list then list [ elif list then list ] ... [ else list ] fi
幾種可能的寫法

---

**第一種：**

```
if list then
   do something here
fi
```

當 list 表述返回值為 True(0)時，將會執行"do something here"。

**範例一：當我們要執行一個命令或程式之前，有時候需要檢查該命令是否存在，然後才執行。**

```
if [ -x /sbin/quotaon ] ; then
      echo "Turning on Quota for root filesystem"
```

```
        /sbin/quotaon /
fi
```

**範例二：當我們將某個檔案做為設定檔時，可先檢查是否存在，然後將該檔案設定值載入。**

```
# Filename : /etc/ppp/settings
PHONE=1-800-COLLECT

#!/bin/sh
# Filename : phonebill
if [ -f /etc/ppp/settings ] ; then
        source /etc/ppp/settings
        echo $PHONE
fi
```

執行

```
[foxman@foxman ppp]# ./phonebill
1-800-COLLECT
```

---

**第二種：**

```
if list then
   do something here
else
   do something else here
fi
```

**範例三：Hostname**

```
#!/bin/sh
if [ -f /etc/HOSTNAME ] ; then
        HOSTNAME=`cat /etc/HOSTNAME`
else
        HOSTNAME=localhost
fi
```

---

**第三種：**

```
if list then
   do something here
```

```
elif list then
   do another thing here
fi
```

**範例四：如果某個設定檔允許有好幾個位置的話，例如 crontab，可利用 if then elif fi 來找尋。**

```
#!/bin/sh
if [ -f /etc/crontab ] ; then
      CRONTAB="/etc/crontab"
elif [ -f /var/spool/cron/crontabs/root ] ; then
      CRONTAB="/var/spool/cron/crontabs/root"
elif [ -f /var/cron/tabs/root ] ; then
      CRONTAB="/var/cron/tabs/root"
fi
export CRONTAB
```

---

**第四種：**

```
if list then
   do something here
elif list then
   do another thing here
else
   do something else here
fi
```

**範例五：我們可利用 uname 來判斷目前系統，並分別做各系統狀況不同的事。**

```
#!/bin/sh
SYSTEM=`uname -s`
if [ $SYSTEM = "Linux" ] ; then
   echo "Linux"
elif [ $SYSTEM = "FreeBSD" ] ; then
   echo "FreeBSD"
elif [ $SYSTEM = "Solaris" ] ; then
   echo "Solaris"
else
   echo "What?"
fi
```

### 6.4.4.5 控制迴圈 - while/until

while list do list done
當 list 為 True 時，該迴圈會不停地執行。
**範例一：無限回圈寫法**

```
#!/bin/sh
while : ; do
   echo "do something forever here"
   sleep 5
done
```

**範例二：強迫把 pppd 殺掉。**

```
#!/bin/sh
while [ -f /var/run/ppp0.pid ] ; do
      killall pppd
done
```

---

until list do list done
當 list 為 False(non-zero)時，該迴圈會不停地執行。
**範例一：等待 pppd 上線。**

```
#!/bin/sh
until [ -f /var/run/ppp0.pid ] ; do
      sleep 1
done
```

### 6.4.4.6 參數與變數

在繼續下去介紹 function 之前，我們必須停下來介紹"參數與變數"。
參數(Parameters)是用來儲存"值"的資料型態，有點像是一般語言中的變數。它可以是個名稱(name)、數字(number)、或者是以下所列出來一些特殊符號(Special Parameters)。
在 shell 中，變數是由 name 形式的參數所構成的。

在前面的許多範例中，我們事實上已經看到許多參數的運用。要設定一個 Parameter 實際很簡單:：
name=value
例如說:
MYHOST="foxman"
而要使用它時，則是加個"$"符號。

echo $MYHOST

---

**位置參數(Positional Parameters)：**

所謂的位置參數便是 0,1,2,3,4,5,6,7,8,9...。使用時，用$0,$1,$2...。
位置參數是當 script 被載入時，後面所附加的參數。$0 是本身，$1 則為第一個參數，$2 為第二個，依此類推。而當 Positional Parameters 被 function 所使用時，它們會被暫時取代(下一節會介紹 function)。

例如以下這個 script:

```
#!/bin/sh
# Filename : position
echo $0
echo $1
```

執行時：

```
[foxman@foxman bash]# ./position abc
./position
abc
```

當位置參數超過兩位數時，有特別的方法來展開，稱為 Expansion。

**特殊參數(Speical Parameters) ：**

這些符號，非常不人性，對新手來說很困擾。但上手後，會覺得方便無比，有些如果您看不懂的話，就--算了，不用浪費太多時間在上面。

---

*** 星號 ：**
將 Positional Parameters 合成一個參數，其間隔為 IFS 內定參數的第一個字元(見內建變數一節)。
**範例：**

```
#!/bin/sh
# starsig
echo $*
```

執行：

```
[root@net53 /root]# starsig a b c d e f g
a b c d e f g
```

---

**@ at 符號 ：**

與*星號類同。不同之處在於不參照 IFS。

**範例：**

```
#!/bin/sh
# atsig
echo $@
```

執行：

```
[root@net53 /root]# atsig a b c d e f g
a b c d e f g
```

---

**# 井字號 ：**

展開 Positional parameters 的數量。

**範例：**

```
#!/bin/sh
# poundsig
echo $#
```

執行

```
[root@net53 /root]# poundsig a b c d e f g
7
```

---

**? 問號 ：**

最近執行的 foreground pipeline 的狀態。

---

**- 減號：**

最近執行的 foreground pipeline 的選項參數。

---

**$ 錢錢錢 ：**

本身的 Process ID。

```
[root@net53 /root]# ps ax | grep bash
1635    p1 S       0:00 /bin/bash

[root@net53 /root]# echo $$
1635
```

---

**! 驚歎號 ：**

最近執行背景命令的 Process ID。

---

**0 零：**

在 Positional Parameters 一部份已經說明過了，是執行的 shell script 本身。但如果是用"bash -c"，則$0 被設為第一個參數。

```
[root@net53 /root]# echo $0
/bin/bash
```

---

**_ 底線符號 ：**

顯示出最後一個執行的命令。

```
[root@net53 /root]# echo $_
bash
```

---

**內建變數(Shell Variables)：**

Bash 有許多內建變數，像 PATH、HOME、ENV.....等等。這些內建變數將在另一節中，專門一一說明。

### 6.4.4.7 函數 – function

[ function ] name () { list; }

function 的參數是 Positional Paraments。

**範例：**

```
#!/bin/sh

function func() {
   echo $1
   echo $2
   return 1
}

func "Hello" "function"
```

局部變數可用 local 來宣告。
函數可 export，使用下一層的 shell 可以使用。
函數可遞迴，沒有遞迴層數的限制。

### 6.4.4.8 bash 內建指令集

其實 bash 也有跟 DOS 的 command 一樣有內建指令，所謂的內建指令是指這個命令屬於 shell 的一部份，因此它不像外部指令一般在執行時要先行載入，所以它執行的速度比較快。以下的命令，大部份都沒有使用範例，您可能會看不出所以然，摸不著頭腦。在我加入範例說明前，建議您"man bash"，然後自己實際操作一次。

---

**[arguments]**

不做任何事，除了[arguments]一些參數展開及一些特定重導向的作業外。永遠返回零。它的用法跟 true 一樣。

---

**filename [arguments]**

source filename [arguments]
由 filename 中讀取命令，並執行。
您會在/etc/rc.d/*中發現很多
. /xxxx
的指令，而 xxxx 的 permission 都不是可執行的。事實上，在 tcsh 中，需要用
source /xxxx
來做同樣的指令。
注意到"."的後面是有空格的(比較一下". /"跟"./"，不一樣)。filename 是內含指令的純文字檔即可，無須 chmod 755 filename。

**範例：**

filename : my_source

```
DEV=lo
IP=127.0.0.1
NETMASK=255.0.0.0
BROADCAST=127.255.255.255
ifconfig $IP netmask $NETMASK broadcast $BROADCAST dev $DEV
```

接下來

```
. my_source
```

或

```
source my_source
```

便可執行該 script，而不需要"chmod 755 my_source"

---

**alias [name[=value] ...]**

暱稱命令

例如您如果來自 DOS 的世界，對 UNIX 的指令不習慣，可用 alias 來修改，以符合您的習慣。

**範例：**

alias ls="ls --color"

alias dir="ls"

alias cd..="cd .."

alias copy="cp -f" # dangerous, recommend, "cp -i"

alias del="rm -f" # dangerous, recommend, "rm -i"

alias move="mv -f" # dangerous, recommend, "mv -i"

alias md="mkdir"

alias rd="rmdir"

---

**unalias [-a] [name ...]**

unalias 取消 alias 的設定。"unalias -a"將全部 alias 取消。

**範例：**

unalias copy

---

**bg [jobspec]**

將指定任務放到背景中，如果 jobspec 未指定，內定為目前的。

---

**fg [jobspec]**

將指定任務放到前景中，如果 jobsepc 沒有指定，那麼內定為目前的。

---

**jobs [-lnp] [ jobspec ... ]**

第一種形式列出目前正在工作的任務。

-l : 除了列出一般資訊外，還列出 Process IDs。

-p : 僅列出該工作群"首腦"(Process group leader)的 Process ID.

-n : 則僅列出有改變的 jobs 的狀態。

如果給定 jobspec，輸出資訊則只有該 jobspec。

返回值為零，除非有非法的選項發生。

jobs -x command [ args ... ]

如果使用第二種形式(-x)，jobs 取代指定的 command 及 args，並執行返回其 Exit Status。

---

**kill [-s sigspec | -sigspec] [pid | jobspec] ...**

將 sigspec 的信號送到 pid 或 jobspec。
sigspec 可以是 SIGKILL/KILL 這種形式或是信號號碼。如果 sigspec 是 signal name，則大小寫無關，而且可以沒有 SIG。
kill -l [signum]
列出信號名稱。

[root@net53 /root]# kill -l

| | | | |
|---|---|---|---|
| 1) SIGHUP | 2) SIGINT | 3) SIGQUIT | 4) SIGILL |
| 5) SIGTRAP | 6) SIGIOT | 7) SIGBUS | 8) SIGFPE |
| 9) SIGKILL | 10) SIGUSR1 | 11) SIGSEGV | 12) SIGUSR2 |
| 13) SIGPIPE | 14) SIGALRM | 15) SIGTERM | 17) SIGCHLD |
| 18) SIGCONT | 19) SIGSTOP | 20) SIGTSTP | 21) SIGTTIN |
| 22) SIGTTOU | 23) SIGURG | 24) SIGXCPU | 25) SIGXFSZ |
| 26) SIGVTALRM | 27) SIGPROF | 28) SIGWINCH | 29) SIGIO |
| 30) SIGPWR | | | |

---

**wait [n]**

等待指定的行程，並返回其結束狀態。n 可以是個 jobspec 或 Process ID。如果 n 未指定，則等待所有的子行程，及返回值為零。若 n 為不存在的 job 或 process，則返回 127。否則，返回值為最後一個 job/process 的 Exit Status。

---

**bind [-m keymap] [-lvd] [-q name]**
**bind [-m keymap] -f filename**
**bind [-m keymap] keyseq:function-name**
顯示出目前 readline 的按鍵及鍵結函數設定或是巨集。
-m keymap : 設定 keymap binding。
-l : 顯示出所有 readline function 的名稱。
-v : 顯示出目前的 function name 及 bindings。
-d : 顯示出 function name 及 bindings。
-f filename : 從 filename 讀取 key bindings。
-q function : 詢問那個按鍵觸發 function。

---

**break [n]**
跳出控制回圈 for/while/until 中使用。如果有指定 n，則跳出 n 層。n 必須是大於等於 1。若 n 大於巢狀迴圈數，則所有的迴圈都會跳離。返回值回零。

---

**continue [n]**

還原控制回圈 for/while/until 中使用。如果有指定 n，則返回 n 層。n 必須是

大於等於 1。若 n 大於巢狀迴圈數，則還原到最上層。返回值回零。

---

**exit [n]**

離開程式。n 是 Exit Status。

---

**return [n]**

在 function 中使用。n 為返回值，其作用與 Exit Status 一樣。

---

**builtin shell-builtin [arguments]**

執行內建函數。當您定義了與內建函數相同的指令時，可用此命令來執行內建函數。

---

**cd [dir]**

更換目錄到 dir。如果沒有指定，內定為 HOME 所指定的目錄。

---

**command [-pVv] command [arg ...]**

用 command 指定可取消正常的 shell function 尋找。只有內建命令及在 PATH 中找得到的才會被執行。"-p"選項，搜尋命令的方式是用 PATH 來找。"-V"或"-v"選項，會顯示出該命令的一些簡約描述。

---

**declare [-frxi] [name[=value]]**

**typeset [-frxi] [name[=value]]**

宣告參數並給它們設定屬性。如果沒有給定名稱，將會顯示各參數值。

-f : 僅使用函數名稱。

-r : 將 name 設為 readonly。

-x : 將 name 輸出給後續環境使用。

-i : 該參數被設為 integer 來使用，可用於算術表述。

用"+"時，關閉該屬性。

---

**dirs [-l] [+/-n]**

顯示目前記憶的目錄。目錄可透過 pushd/popd 來操作。

+n : 顯示開始的記錄 n 個。

-n : 顯示結尾的記錄 n 個。

-l : 顯示較多的資訊。

---

**echo [-neE] [arg ...]**

輸出顯示 args，由空白分隔。返回值永為零。

-n : 不跳行。

-e : 啟動"\"符號的解譯。
-E : 將 ESC 解譯功能取消。
"\a" : alert(bell)，發出聲響。
"\b" : backspace，倒退。
"\c" : suppress trailing newline，不跳行。
"\f" : form feed，跳行跳格。
"\n" : new line，新行。
"\r" : carriage return，回到行起點。
"\t" : horizontal tab，水平跳位。
"\v" : vertical tab，垂直跳位。
"\\" : 輸出"\"。
"\nnn" : 輸出 ASCII Code 號碼 nnn(八進位)。

---

**enable [-n] [-all] [name ...]**

啟動或關閉內建函數命令。使用"-n"將所有指定命令皆關閉，否則都是啟動的。如果只有"-n"參數，它將會顯示所有關閉的函數。如果只有"-all"，它將會顯示所有內建命令。

---

**eval [arg ...]**

讀取 args，並將 args 合為一個命令，然後執行。其返回值成為 eval 的返回值。如果沒有參數，eval 返回 True。

---

**exec [[-] command [arguments]]**

當命令執行時，該命令取代 shell，沒有新的 process 產生。如果第一個參數是"-"，shell 會將"-"放入第零個參數，傳給 command。

---

**export [-nf] [name[=word]] ...**

export -p

將 name 輸出給環境，給往後的命令使用。"-f"選項表示 name 是函數。"-p"顯示出所有 export 的名稱。"-n"移除 name。

---

**set [--abefhkmnptuvxldCHP] [-o option] [arg ...]**

-a : 自動將變數標記為可讓後面環境所使用。
-b : 立即報告被終結的背景程式狀態。
-e : 當命令(simple-command，見後面)返回非零值時，立即跳出。
-f : 取消 pathname expansion。
-h : 找出所記憶的函數命令位置。

-k : 所有 keyword 參數都放到環境中。
-m : 監督模式。
-n : 讀取命令，但不要執行。可用於語法檢查。
-p : 打開 privileged 模式。
-t : 當讀取一個命令並執行後，立即離開。
-u : 當參數展開時，把 unset 參數當成是錯誤。
-v : 列出 shell input lines。
-x : 在展開每個 simple-command 後，bash 顯示展開值在 PS4 上。
-l : 儲存並還原 name binding 在 for 語法中。
-d : 關閉 hasing command 搜尋。
-C : 跟`noclobber=`一樣。請見內定參數一節。
-H : 啟動! style history substitution。
-P : 在使用像 cd 這種指令時，不要跟隨 symbolic links。
-- : "--"之後，沒有參數跟在後面。
- : 指定將所有後面的參數當成是位置參數。
-o option-name : option-name 可以是以下之一
allexport : 與"-a"相同。
braceexpand : 啟動 Brace Expansion。這是內定設定。
emacs : 使用 emacs-style 命令列編輯界面。
errexit : 與"-e"相同。
histexpand : 與"-H"相同。
ignoreeof : 效果跟`IGNOREEOF=10`一樣。
interactive-commands : 允許#做為註解。
monitor : 與"-m"相同。
noclobber : 與"-C"相同。
noexec : 與"-n"相同。
noglob : 與"-f"相同。
nohash : 與"-d"相同。
notify : 與"-b"相同。
nounset : 與"-u"相同。
physical : 與"-P"相同。
posix : Bash 行為修改為 Posix 1003.2 標準。
privileged : 與"-p"相同。
verbose : 與"-v"相同。
vi : 使用 vi-style 命令列編輯程式。
xtrace : 與"-x"相同。

---

**unset [-fv] [name ...]**

移除對映於 name 的參數。要注意 PATH、IFS、PPID、PS1、PS2、UID、EUID 不能 unset。若 RANDOM、SECONDS、LINENO、HISTCMD 被 unset，它們會喪失原有意義，既始它們後來被重設也一樣。返回值為 True，除非 name 是不能被 unset 的。

---

**fc [-e ename] [-nlr] [first] [last]**

**fc -s [pat=rep] [cmd]**

修正命令。

---

**getopts optstring name [args]**

解析位置參數。

---

**hash [-r] [name]**

對每個 name 命令的完整路徑記錄下來。"-r"選項強迫忘記所有命令位置。如果沒有給參數，則將會印出所有的資訊。返回值為 True。

---

**help [pattern]**

顯示協助資訊。

---

**history [n]**

**history -rwan [filename]**

沒有參數時，會顯示所下命令的歷史記錄。帶有參數"n"則顯示最後 n 個。

其它參數如下：

-a：新增"新歷史"到歷史檔中。

-n：讀取尚未讀到歷史中的記錄。

-r：讀取 filename 做為歷史檔，並用它為目前歷史記錄。

-w：將現有歷史記錄寫到 filename 中。

---

**let arg [arg ...]**

算術表述。請參考算術表述一節。

---

**local [name[=value] ...]**

產生一個局部參數。如果用於 function，則其作用範圍在 function 內及其子函數。

---

**logout**

離開 login shell。

---

**popd [+/-n]**

移除目錄堆疊。"+n"移除上面 n 個，"-n"移除下面 n 個。

---

**pushd [dir]**

**pushd +/-n**

將目錄新增到目錄堆疊的最上面。"+n"旋轉該堆疊，使第 n 個目錄變成最上面。"-n"旋轉該堆疊，使倒數第 n 個目錄變成最上面。

---

**pwd**

列出目前工作目錄的絕對路徑。

---

**read [-r] [name ...]**

讀進一行，然後第一個字設到第一個 name，第二個設到第二個 name，依此類推。如果沒有 name 在參數中，則 read 會將值設到 REPLY。返回值為零，除非遇到 End-Of-File。若有"-r"選項，則"\n"被考慮為該行的一部份。

---

**readonly [-f] [name ...]**

**readonly -p**

將給定的 name 標記為 readonly。如果是"-f"選項，則函數也一樣被標記為 readonly。"-p"會列出所有 readonly 的 name。"--"取消檢查剩餘的參數。

---

**shift [n]**

Positional Parameters 從 n+1...開始，會被改為$1...。n 若為零，則沒有改變。n 若未給定，則內定為 1。n 必須是非負數，並且小於或等於$#。若 n 大於$#，則沒有改變。返回值為零，除非 n 大於$#或小於零。

---

**suspend [-f]**

暫停這個 shell 的執行，直到它收到 SIGCONT 信號。"-f"選項則是叫 login shell 不要抱怨，不過還是一樣暫停。返回狀態零，除非該 shell 是個 login shell，而且沒有"-f"選項。

---

**test expr**

**[ expr ]**

我們在 Exit Status 的部份已經說過了，不再重複。

---

**times**

列出該 shell 的累積的使用者及系統時間及從 shell 執行的 process 時間，返回值為零。

---

**trap [-l] [arg] [sigspec]**
當收到 sigspec 信號時，執行 arg 命令。"-l"顯示出信號名稱及號碼。

---

**type [-all] [-type | -path] name [name ...]**

沒有參數的狀況下，它會顯示出 shell 如何解譯 name 做為命令。如果有"-type"，它將會顯示 alias、keyword、function、builtin 或 file。如果有"-path"的參數，它將會顯示該命令的路徑，找不到的話，不顯示任何東西。如果有"-all"的參數，它將會顯示所有可執行 name 的可能路徑。type 接受"-a"、"-t"、"-p"做為縮寫。

---

**ulimit [-SHacdfmstpnuv [limit]]**
ulimit 提供了對 shell 的可獲取資源控制的功能。
-a：報告目前所有限制。
-c：設定最大可產生的 core 檔案。
-d：行程資料段(process's data segment)最大值。
-f：可被這個 shell 產生的最大檔案。
-m : resident set size 最大值。
-s：堆疊最大值。
-t : CPU TIME 最大值(以秒計算)。
-p : pipe size in 512-byte blocks 的最大值。
-n：可開啟的 file descriptors 最大值。
-u：單一使用者可使用的最大 process 數。
-v：該 shell 最大虛擬記憶體可用值。
所有項目是以 1024 做為單位。

---

umask [-S] [mode]
將使用者的 file-creation mask 設為 mode。"-S"選項將 mask 印成符號形式。

3.4.4.9 bash 內建參數

Bash 內建參數

---

PPID：該 bash 的呼叫者 process ID.
PWD：目前的工作目錄。
OLDPWD：上一個工作目錄。
REPLY：當 read 命令沒有參數時，直接設在 REPLY 上。
UID : User ID。
EUID : Effective User ID。

BASH : Bash 的完整路徑。
BASH_VERSION : Bash 版本。
SHLVL : 每次有 Bash 執行時，數字加一。
RANDOM : 每次這個參數被用到時，就會產生一個亂數在 RANDOM 上。
SECONDS : 從這個 Shell 一開始啟動後的時間。
LINENO : Script 的行數。
HISTCMD : 歷史記錄數。
OPTARG : getopts 處理的最後一個選項參數。
OPTIND : 下一個要由 getopts 所處理的參數號碼。
HOSTTYPE : 機器種類。
OSTYPE : 作業系統名稱。
IFS : Internal Field Separator。
PATH : 命令搜尋路徑。 PATH="/usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin:."
HOME : 目前使用者的 home directory;
CDPATH : cd 命令的搜尋路徑。
ENV : 如果這個參數被設定，每次有 shell script 被執行時，將會執行它所設定的檔名做為環境設定。
MAIL : 如果這個參數被設定，而且 MAILPATH 沒有被設定，那麼有信件進來時，bash 會通知使用者。
MAILCHECK : 設定多久時間檢查郵件一次。
MAILPATH : 一串的郵件檢查路徑。
MAIL_WARNING : 如果有設定的話，郵件被讀取後，將會顯示訊息。
PS1 : 提示訊息設定，內定為"bash\$ "。(請詳見提示訊息一節。)
PS2 : 第二提示訊息設定，內定為"> "。
PS3 : select 命令所使用的提示訊息。
PS4 : 執行追蹤時用的提示訊息設定，內定為"+ "。
HISTSIZE : 命令歷史記錄量，內定為 500。
HISTFILE : 歷史記錄檔，內定~/.bash_history。
HISTFILESIZE : 歷史記錄檔行數最大值，內定 500。
OPTERR : 如果設為 1，bash 會顯示 getopts 的錯誤。
PROMPT_COMMAND : 如果設定的話，該值會在每次執行命令前都顯示。
IGNOREEOF : 將 EOF 值當成輸入，內定為 10。
TMOUT : 如果設為大於零，該值被解譯為輸入等待秒數。若無輸入，當成沒有輸入。
FCEDIT : fc 命令的內定編輯器。
FIGNORE : 請詳見 READLINE。
INPUTRC : readline 的 startup file，內定~/.inputrc
notify : 如果設定了，bash 立即報告被終結的背景程式。

history_control, HISTCONTROL : history 使用。
command_oriented_history : 存入多行指令。
glob_dot_filenames : 如果設定了，bash 將會把"."包含入檔案路徑中。
allow_null_glob_expansion : 如果設定了，bash 允許路徑明稱為 null string。
histchars : history 使用。
nolinks : 如果設定了，執行指令時，不會跟隨 symbolic links。
hostname_completion_file, HOSTFILE : 包含與/etc/hosts 相同格式的檔名。
noclobber : 如果設定了，Bash 不會覆寫任何由">"、">&"及"<>"所操作的檔案。
auto_resume : 請見任務控制一節。
no_exit_on_failed_exec : 如果該值存在，非互動的 shell 不會因為 exec 失敗而跳出。
cdable_vars : 如果啟動，而 cd 命令找不到目錄，可切換到參數形態指定的目錄下。

#### 6.4.4.10 提示符號

**提示符號 ：**
Bash 使用 PS1~PS4 來顯示提示符號，其格式如下：
\t : 現在時間。
\d : 現在日期。
\n : 新行。
\s : shell 的名稱。
\w : 目前工作目錄。
\W : 目前工作目錄完整路徑。
\u : 使用者名稱。
\h : Hostname。
\# : 這個命令的號碼。
\! : 歷史號碼。
\$ : 如果 EUID 是 0，則#，否則為$。
\nnn : 八進位的字元。
\\ : "\"符號。
\[ : 開始一序列不可列印的字元。
\] : 結束一序列不可列印的字元。

#### 6.4.4.11 算術優先等級

**算術優先等級：**
---

- + （正負）
! ~
* / %
+ -
<< >>
<= >= < >
== !=
&
^
|
&&
||
= *= /= %= += -= <<= >>= &= ^= |=

#### 6.4.4.12 重導 Redirection

重導之前有稍微介紹過，熟悉嗎？
**重導 Redirection**
---
> 轉向輸出
< 轉向輸入
>> 附加輸出
2> 轉向錯誤
1>&2 轉向輸出跟隨著錯誤
2>&1 轉向錯誤跟隨著輸出

## 6.5 Linux 的登出

當您使用 Linux 一段時間，想要休息一下或者先去吃個飯，記得要先 "登出" 系統，可不要放著系統在提示符號底下就跑掉了唷，因為這樣一來，任何人都可以以您的帳號對電腦為所欲為，等於是放任自己的系統任人宰割，完全失去了安全性，小心有人趁機冒用您的帳號寫電子郵件發表不當的言論，到時候可是百口莫辯囉。

登出 Linux 系統方法有三種：
1. 鍵入 logout
2. 鍵入 exit
3. 按【Ctrl】+【D】鍵

在你順利登出系統後，便會出現登入的歡迎畫面，來讓下一位使用者登入。

## 6.6 Linux 的關機

對任何一部多人多工的主機來說，正確的關機程序是非常的重要的。設法記住一個原則，要關機的時後，不要隨便就關掉電源，非正常的中斷電源可能會造成檔案系統損毀，雖然說 Linux 會自己設法修復毀損的檔案系統，但難保事情不會出差錯。

執行關機的方法有好幾種，上一節的方法也是，但標準的方法是用 shutdown 指令，這指令會發出警告訊息給線上的所有使用者，經過您所設定的時間後，會自動刪除所有工作，並 umount 所有的檔案系統，回到純單人使用模式，此時您就可以關掉電源。

shutdown 常用語法：【shutdown 「參數」『時間』『給使用者的訊息』】

shutdown 指令有許多參數，介紹如下：

1. –r ： shutdown –r 就如同 reboot 一樣，只是您可以指定多久以後重新開機。

2. –h ： shutdown –h 就如同 halt 一樣，只是您可以指定多久後才關閉系統。

3. –k ： shutdown –k 這只是通知正在線上的所有使用者，系統即將關機，但不會真的關機，系統管理者必須要再下 reboot 指令才會重新啟動。但要注意的是，shutdown –k 會建立 /etc/nologin 這個檔案，而導致使用者無法再登陸系統，如果不想重新啟動，要記得移除 /etc/nologin。

4. 『時間』格式：
   now：表示現在，馬上、立刻的意思，這是最常用的。
   + 數字：單位為幾分鐘，如 +1 表示 1 分鐘以後關機。

在下例中，系統會廣播” Please logoff in a minute , HURRY! “訊息，經過一分鐘後，系統會解除所有的背景工作，以及強迫趕出所有使用者，之後就回到單人使用狀態，此時就可切斷電源。

```
[root@net53 /root]# shutdown    –r    +1 ‘ Please logoff in a minute , HURRY! ‘
```