鳥哥的 Linux 私房菜

為取得較佳瀏覽結果，請愛用 firefox 瀏覽本網頁

第十一章、認識與學習 BASH

切換解析度為 800x600

最近更新日期：2009/08/25

在 Linux 的環境下，如果你不懂 bash 是什麼，那麼其他的東西就不用學了！因為前面幾章我們使用終端機下達指令的方式， 就是透過 bash 的環境來處理的喔！所以說，他很重要吧！bash 的東西非常的多，包括變數的設定與使用、bash 操作環境的建置、資料流重導向的功能，還有那好用的管線命令！好好清一清腦門，準備用功去囉～ ^_^ 這個章節幾乎是所有指令列模式（command line）與未來主機維護與管理的重要基礎，一定要好好仔細的閱讀喔！

認識 BASH 這個 Shell

我們在第一章 Linux 是什麼當中提到了： 管理整個電腦硬體的其實是作業系統的核心 (kernel)，這個核心是需要被保護的！ 所以我們一般使用者就只能透過 shell 來跟核心溝通，以讓核心達到我們所想要達到的工作。 那麼系統有多少 shell 可用呢？為什麼我們要使用 bash 啊？底下分別來談一談喔！

硬體、核心與 Shell

這應該是個蠻有趣的話題：『什麼是 Shell 』？相信只要摸過電腦，對於作業系統 (不論是 Linux 、Unix 或者是 Windows) 有點概念的朋友們大多聽過這個名詞，因為只要有『作業系統』那麼就離不開 Shell 這個東西。不過，在討論 Shell 之前，我們先來瞭解一下電腦的運作狀況吧！ 舉個例子來說：當你要電腦傳輸出來『音樂』的時候，你的電腦需要什麼東西呢？

1. 硬體：當然就是需要你的硬體有『音效卡晶片』這個配備，否則怎麼會有聲音；
2. 核心管理：作業系統的核心可以支援這個晶片組，當然還需要提供晶片的驅動程式囉；
3. 應用程式：需要使用者 (就是你) 輸入發生聲音的指令囉！

這就是基本的一個輸出聲音所需要的步驟！也就是說，你必須要『輸入』一個指令之後， 『硬體』才會透過你下達的指令來工作！那麼硬體如何知道你下達的指令呢？那就是 kernel (核心) 的控制工作了！也就是說，我們必須要透過『 Shell 』將我們輸入的指令與 Kernel 溝通，好讓 Kernel 可以控制硬體來正確無誤的工作！ 基本上，我們可以透過底下這張圖來說明一下：

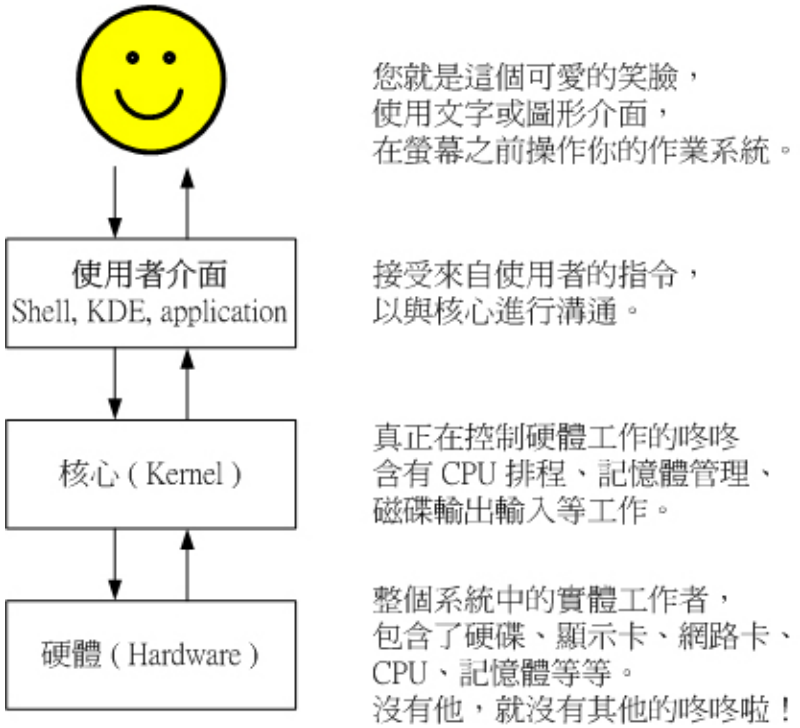

圖 1.1.1、硬體、核心與使用者的相關性圖示

我們在第零章內的作業系統小節曾經提到過， 作業系統其實是一組軟體，由於這組軟體在控制整個硬體與管理系統的活動監測， 如果這組軟體能被使用者隨意的操作，若使用者應用不當，將會使得整個系統崩潰！因為作業系統管理的就是整個硬體功能嘛！ 所以當然不能夠隨便被一些沒有管理能力的終端用戶隨意使用囉！

但是我們總是需要讓使用者操作系統的，所以就有了在作業系統上面發展的應用程式啦！使用者可以透過應用程式來指揮核心， 讓核心達成我們所需要的硬體任務！如果考慮如第零章所提供的作業系統圖示(圖4.2.1)， 我們可以發現應用程式其實是在最外層，就如同雞蛋的外殼一樣，因此這個咚咚也就被稱呼為殼程式 (shell) 囉！

其實殼程式的功能只是提供使用者操作系統的一個介面，因此這個殼程式需要可以呼叫其他軟體才好。 我們在第五章到第十章提到過很多指令，包括 man, chmod, chown, vi, fdisk, mkfs 等等指令，這些指令都是獨立的應用程式， 但是我們可以透過殼程式 (就是指令列模式) 來操作這些應用程式，讓這些應用程式呼叫核心來運作所需的工作哩！ 這樣對於殼程式是否有了一定的概念了？

Tips:
也就是說，只要能夠操作應用程式的介面都能夠稱為殼程式。狹義的殼程式指的是指令列方面的軟體，包括本章要介紹的 bash 等。 廣義的殼程式則包括圖形介面的軟體！因為圖形介面其實也能夠操作各種應用程式來呼叫核心工作啊！ 不過在本章中，我們主要還是在使用 bash 啦！

為何要學文字介面的 shell？

文字介面的 shell 是很不好學的，但是學了之後好處多多！所以， 在這裡鳥哥要先對您進行一些心理建設，先來瞭解一下為啥學習 shell 是有好處的，這樣你才會有信心繼續玩下去 ^_^

- **文字介面的 shell：大家都一樣！**

鳥哥常常聽到這個問題：『我幹嘛要學習 shell 呢？ 不是已經有很多的工具可以提供我設定我的主機了？我為何要花這麼多時間去學指令呢？不是以 X Window 按一按幾個按鈕就可以搞定了嗎？』唉～還是得一再地強調， X Window 還有 Web 介面的設定工具例如 Webmin (註1) 是真的好用的傢伙，他真的可以幫助我們很簡易的設定好我們的主機，甚至是一些很進階的設定都可以幫我們搞定。

但是鳥哥在前面的章節裡面也已經提到過相當多次了， X Window 與 web 介面的工具，他的介面雖然親善，功能雖然強大， 但畢竟他是將所有利用到的軟體都整合在一起的一組應用程式而已， 並非是一個完整的套件，所以某些時候當你升級或者是使用其他套件管理模組 (例如 tarball 而非 rpm 檔案等等) 時，就會造成設定的困擾了。甚至不同的 distribution 所設計的 X window 介面也都不相同，這樣也造成學習方面的困擾。

文字介面的 shell 就不同了！幾乎各家 distributions 使用的 bash 都是一樣的！如此一來， 你就能夠輕輕鬆鬆的轉換不同的 distributions， 就像武俠小說裡面提到的『一法通、萬法通！』

- **遠端管理：文字介面就是比較快！**

此外，Linux 的管理常常需要透過遠端連線，而連線時文字介面的傳輸速度一定比較快， 而且，較不容易出現斷線或者是資訊外流的問題，因此，shell 真的是得學習的一項工具。而且，他可以讓您更深入 Linux， 更瞭解他，而不是只會按一按滑鼠而已！所謂『天助自助者！』多摸一點文字模式的東西，會讓你與 Linux 更親近呢！

- **Linux 的任督二脈： shell 是也！**

有些朋友也很可愛，常會說：『我學這麼多幹什麼？ 又不常用，也用不到！』嘿嘿！有沒有聽過『書到用時方恨少？』當你的主機一切安然無恙的時候，您當然會覺得好像學這麼多的東西一點幫助也沒有呀！ 萬一，某一天真的不幸給他中標了，您該如何是好？是直接重新安裝？ 還是先追蹤入侵來源後進行漏洞的修補？或者是乾脆就關站好了？這當然涉及很多的考量， 但就以鳥哥的觀點來看，多學一點總是好的，尤其我們可以有備而無患嘛！甚至學的不精也沒有關係，瞭解概念也就 OK 啦！畢竟沒有人要您一定要背這麼多的內容啦！瞭解概念就很了不起了！

此外，如果你真的有心想要將您的主機管理的好，那麼良好的 shell 程式編寫是一定需要的啦！就鳥哥自己來說，鳥哥管理的主機雖然還不算多， 只有區區不到十部，但是如果每部主機都要花上幾十分鐘來查閱他的登錄檔資訊以及相關的訊息， 那麼鳥哥可能會瘋掉！基本上，也太沒有效率了！這個時候，如果能夠藉由 shell 提供的資料流重導向以及管線命令，呵呵！ 那麼鳥哥分析登錄資訊只要花費不到十分鐘就可以看完所有的主機之重要資訊了！相當的好用呢！

由於學習 shell 的好處真的是多多啦！所以，如果你是個系統管理員，或者有心想要管理系統的話，那麼 shell 與 shell scripts 這個東西真的有必要看一看！因為他就像『打通任督二脈，任何武功都能隨你應用』的說！

系統的合法 shell 與 /etc/shells 功能

知道什麼是 Shell 之後，那麼我們來瞭解一下 Linux 使用的是哪一個 shell 呢？什麼！哪一個？難道說 shell 不就是『一個 shell 嗎？』哈哈！那可不！由於早年的 Unix 年代，發展者眾，所以由於 shell 依據發展者的不同就有許多的版本，例如常聽到的 Bourne SHell (sh) 、在 Sun 裡頭預設的 C SHell、 商業上常用的 K SHell、, 還有 TCSH 等等，每一種 Shell 都各有其特點。至於 Linux 使用的這一種版本就稱為『Bourne Again SHell (簡稱 bash)』，這個 Shell 是 Bourne Shell 的增強版本，也是基準於 GNU 的架構下發展出來的呦！

在介紹 shell 的優點之前，先來說一說 shell 的簡單歷史吧(註2)：第一個流行的 shell 是由 Steven Bourne 發展出來的，為了紀念他所以就稱為 Bourne shell，或直接簡稱為 sh ！而後來另一個廣為流傳的 shell 是由柏克萊大學的 Bill Joy 設計依附於 BSD 版的 Unix 系統中的 shell，這個 shell 的語法有點類似 C 語言，所以才得名為 C shell，簡稱為 csh ！由於在學術界 Sun 主機勢力相當的龐大，而 Sun 主要是 BSD 的分支之一，所以 C shell 也是另一個很重要而且流傳很廣的 shell 之一 。

Tips:
由於 Linux 為 C 程式語言撰寫的，很多程式設計師使用 C 來開發軟體，因此 C shell 相對的就很熱門了。 另外，還記得我們在第一章、Linux 是什麼提到的吧？ Sun 公司的創始人就是 Bill Joy，而 BSD 最早就是 Bill Joy 發展出來的啊。

那麼目前我們的 Linux (以 CentOS 5.x 為例) 有多少我們可以使用的 shells 呢？ 你可以檢查一下 /etc/shells 這個檔案，至少就有底下這幾個可以用的 shells：

- /bin/sh (已經被 /bin/bash 所取代)
- /bin/bash (就是 Linux 預設的 shell)
- /bin/ksh (Kornshell 由 AT&T Bell lab. 發展出來的，相容於 bash)
- /bin/tcsh (整合 C Shell， 提供更多的功能)
- /bin/csh (已經被 /bin/tcsh 所取代)
- /bin/zsh (基於 ksh 發展出來的，功能更強大的 shell)

雖然各家 shell 的功能都差不多，但是在某些語法的下達方面則有所不同，因此建議你還是得要選擇某一種 shell 來熟悉一下較佳。 Linux 預設就是使用 bash， 所以最初你只要學會 bash 就非常了不起了！^_^！ 另外，咦！為什麼我們系統上合法的 shell 要寫入 /etc/shells 這個檔案啊？ 這是因為系統某些服務在運作過程中，會去檢查使用者能夠使用的 shells， 而這些 shell 的查詢就是藉由 /etc/shells 這個檔案囉！

舉例來說，某些 FTP 網站會去檢查使用者的可用 shell， 而如果你不想要讓這些使用者使用 FTP 以外

的主機資源時，可能會給予該使用者一些怪怪的 shell，讓使用者無法以其他服務登入主機。 這個時候，你就得將那些怪怪的 shell 寫到 /etc/shells 當中了。舉例來說，我們的 CentOS 5.x 的 /etc/shells 裡頭就有個 /sbin/nologin 檔案的存在，這個就是我們說的怪怪的 shell 囉～

那麼，再想一想，我這個使用者什麼時候可以取得 shell 來工作呢？還有， 我這個使用者預設會取得哪一個 shell 啊？還記得我們在第五章的在終端介面登入linux小節當中提到的登入動作吧？ 當我登入的時候，系統就會給我一個 shell 讓我來工作了。 而這個登入取得的 shell 就記錄在 /etc/passwd 這個檔案內！這個檔案的內容是啥？

```
[root@www ~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....(底下省略).....
```

如上所示，在每一行的最後一個資料，就是你登入後可以取得的預設的 shell 啦！那你也會看到， root 是 /bin/bash ， 不過， 系統帳號 bin 與 daemon 等等，就使用那個怪怪的 /sbin/nologin 囉～關於使用者這部分的內容，我們留在第十四章的帳號管理時提供更多的說明。

Bash shell 的功能

既然 /bin/bash 是 Linux 預設的 shell ， 那麼總是得瞭解一下這個玩意兒吧！bash 是 GNU 計畫中重要的工具軟體之一，目前也是 Linux distributions 的標準 shell 。 bash 主要相容於 sh ， 並且依據一些使用者需求，而加強的 shell 版本。不論你使用的是那個 distribution ， 你都難逃需要學習 bash 的宿命啦！那麼這個 shell 有什麼好處，幹嘛 Linux 要使用他作為預設的 shell 呢？ bash 主要的優點有底下幾個：

- **命令編修能力 (history)：**

bash 的功能裡頭，鳥哥個人認為相當棒的一個就是『他能記憶使用過的指令！』 這功能真的相當的棒！因為我只要在指令列按『上下鍵』就可以找到前/後一個輸入的指令！而在很多 distribution 裡頭，預設的指令記憶功能可以到達 1000 個！也就是說， 你曾經下達過的指令幾乎都被記錄下來了。

這麼多的指令記錄在哪裡呢？在你的家目錄內的 .bash_history 啦！ 不過， 需要留意的是， ~/.bash_history 記錄的是前一次登入以前所執行過的指令， 而至於這一次登入所執行的指令都被暫存在記憶體中，當你成功的登出系統後，該指令記憶才會記錄到 .bash_history 當中！

這有什麼功能呢？最大的好處就是可以『查詢曾經做過的舉動！』 如此可以知道你的執行步驟，那麼就可以追蹤你曾下達過的指令，以作為除錯的工具！ 但如此一來也有個煩惱，就是如果被駭客入侵了，那麼他只要翻你曾經執行過的指令， 剛好你的指令又跟系統有關 (例如直接輸入 MySQL 的密碼在指令列上面)，那你的主機可就傷腦筋了！ 到底記錄指令的數目越多還是越少越好？這部份是見仁見智啦，沒有一定的答案的。

- **命令與檔案補全功能：（[tab] 按鍵的好處）**

還記得我們在第五章內的重要的幾個熱鍵小節當中提到的 [tab] 這個按鍵嗎？這個按鍵的功能就是在 bash 裡頭才有的啦！常常在 bash 環境中使用 [tab] 是個很棒的習慣喔！因為至少可以讓你 1)少打很多字； 2)確定輸入的資料是正確的！ 使用 [tab] 按鍵的時機依據 [tab] 接在指令後或參數後而有所不同。我們再複習一次：

  - [Tab] 接在一串指令的第一個字的後面，則為命令補全；
  - [Tab] 接在一串指令的第二個字以後時，則為『檔案補齊』！

所以說，如果我想要知道我的環境中，所有可以執行的指令有幾個？ 就直接在 bash 的提示字元後面連續按兩次 [tab] 按鍵就能夠顯示所有的可執行指令了。 那如果想要知道系統當中所有以 c 為開頭的指令呢？就按下『 c[tab][tab] 』就好啦！ ^_^

是的！真的是很方便的功能，所以，有事沒事，在 bash shell 底下，多按幾次 [tab] 是一個不錯的習慣啦！

- **命令別名設定功能：（alias）**

假如我需要知道這個目錄底下的所有檔案 (包含隱藏檔) 及所有的檔案屬性，那麼我就必須要下達『 ls -al 』這樣的指令串，唉！真麻煩，有沒有更快的取代方式？呵呵！就使用命令別名呀！例如鳥哥最喜歡直接以 lm 這個自訂的命令來取代上面的命令，也就是說， lm 會等於 ls -al 這樣的一個功能，嘿！那麼要如何作呢？就使用 alias 即可！你可以在指令列輸入 alias 就可以知道目前的命令別名有哪些了！也可以直接下達命令來設定別名呦：

```
alias lm='ls -al'
```

- **工作控制、前景背景控制：（job control, foreground, background）**

這部分我們在第十七章 Linux 程序控制中再提及！ 使用前、背景的控制可以讓工作進行的更為順利！至於工作控制(jobs)的用途則更廣， 可以讓我們隨時將工作丟到背景中執行！而不怕不小心使用了 [Ctrl] + c 來停掉該程序！真是好樣的！此外，也可以在單一登入的環境中，達到多工的目的呢！

- **程式化腳本：（shell scripts）**

在 DOS 年代還記得將一堆指令寫在一起的所謂的『批次檔』吧？在 Linux 底下的 shell scripts 則發揮更為強大的功能，可以將你平時管理系統常需要下達的連續指令寫成一個檔案， 該檔案並且可以透過對談互動式的方式來進行主機的偵測工作！也可以藉由 shell 提供的環境變數及相關指令來進行設計，哇！整個設計下來幾乎就是一個小型的程式語言了！該 scripts 的功能真的是超乎我的想像之外！以前在 DOS 底下需要程式語言才能寫的東西，在 Linux 底下使用簡單的 shell scripts 就可以幫你達成了！真的厲害！這部分我們在第十三章再來談！

- **萬用字元：（Wildcard）**

除了完整的字串之外， bash 還支援許多的萬用字元來幫助使用者查詢與指令下達。 舉例來說，想要知道 /usr/bin 底下有多少以 X 為開頭的檔案嗎？使用：『 ls -l /usr/bin/X* 』就能夠知道囉～此外，還有其他可供利用的萬用字元， 這些都能夠加快使用者的操作呢！

Bash shell 的內建命令： type

我們在第五章提到關於 Linux 的線上說明文件部分，也就是 man page 的內容，那麼 bash 有沒有什麼說明文件啊？開玩笑～ 這麼棒的東西怎麼可能沒有說明文件！請你在 shell 的環境下，直接輸入 man bash 瞧一瞧， 嘿嘿！不是蓋的吧！讓你看個幾天幾夜也無法看完的 bash 說明文件，可是很詳盡的資料啊！ ^_^

不過，在這個 bash 的 man page 當中，不知道你是否有察覺到，咦！ 怎麼這個說明文件裡面有其他的檔案說明啊？舉例來說，那個 cd 指令的說明就在這個 man page 內？ 然後我直接輸入 man cd 時，怎麼出現的畫面中，最上方竟然出現一堆指令的介紹？這是怎麼回事？ 為了方便 shell 的操作，其實 bash 已經『內建』了很多指令了，例如上面提到的 cd， 還有例如 umask 等等的指令，都是內建在 bash 當中的呢！

那我怎麼知道這個指令是來自於外部指令(指的是其他非 bash 所提供的指令) 或是內建在 bash 當中的呢？ 嘿嘿！利用 type 這個指令來觀察即可！舉例來說：

```
[root@www ~]# type [-tpa] name
選項與參數：
    ：不加任何選項與參數時，type 會顯示出 name 是外部指令還是 bash 內建指令
-t  ：當加入 -t 參數時，type 會將 name 以底下這些字眼顯示出他的意義：
      file    ：表示為外部指令；
      alias   ：表示該指令為命令別名所設定的名稱；
      builtin ：表示該指令為 bash 內建的指令功能；
-p  ：如果後面接的 name 為外部指令時，才會顯示完整檔名；
-a  ：會由 PATH 變數定義的路徑中，將所有含 name 的指令都列出來，包含 alias

範例一：查詢一下 ls 這個指令是否為 bash 內建？
[root@www ~]# type ls
ls is aliased to `ls --color=tty' <==未加任何參數，列出 ls 的最主要使用情況
[root@www ~]# type -t ls
alias                             <==僅列出 ls 執行時的依據
[root@www ~]# type -a ls
ls is aliased to `ls --color=tty' <==最先使用 aliase
ls is /bin/ls                     <==還有找到外部指令在 /bin/ls

範例二：那麼 cd 呢？
[root@www ~]# type cd
cd is a shell builtin             <==看到了嗎？ cd 是 shell 內建指令
```

透過 type 這個指令我們可以知道每個指令是否為 bash 的內建指令。 此外，由於利用 type 搜尋後面的名稱時，如果後面接的名稱並不能以執行檔的狀態被找到， 那麼該名稱是不會被顯示出來的。也就是說， type 主要在找出『執行檔』而不是一般檔案檔名喔！ 呵呵！所以，這個 type 也可以用來作為類似 which 指令的用途啦！找指令用的！

指令的下達

我們在第五章的開始下達指令小節已經提到過在 shell 環境下的指令下達方法，如果你忘記了請回到第五章再去回憶一下！這裡不重複說明了。 鳥哥這裡僅就反斜線 (\) 來說明一下指令下達的方式囉！

```
範例：如果指令串太長的話，如何使用兩行來輸出？
[vbird@www ~]# cp /var/spool/mail/root /etc/crontab \
> /etc/fstab /root
```

上面這個指令用途是將三個檔案複製到 /root 這個目錄下而已。不過， 因為指令太長， 於是鳥哥就利用『 \[Enter] 』來將 [Enter] 這個按鍵『跳脫！』開來，讓 [Enter] 按鍵不再具有『開始執行』的功能！好讓指令可以繼續在下一行輸入。 需要特別留意， [Enter] 按鍵是緊接著反斜線（\）的，兩者中間沒有其他字元。 因為 \ 僅跳脫『緊接著的下一個字符』而已！所以，萬一我寫成： 『 \ [Enter] 』， 亦即 [Enter] 與反斜線中間有一個空格時，則 \ 跳脫的是『空白鍵』而不是 [Enter] 按鍵！這個地方請再仔細的看一遍！很重要！

如果順利跳脫 [Enter] 後，下一行最前面就會主動出現 > 的符號， 你可以繼續輸入指令囉！也就是說，那個 > 是系統自動出現的，你不需要輸入。

總之，當我們順利的在終端機 (tty) 上面登入後， Linux 就會依據 /etc/passwd 檔案的設定給我們一個 shell (預設是 bash)，然後我們就可以依據上面的指令下達方式來操作 shell， 之後，我們就可以透過 man 這個線上查詢來查詢指令的使用方式與參數說明， 很不錯吧！那麼我們就趕緊更進一步來操作

bash 這個好玩的東西囉！

Shell 的變數功能

變數是 bash 環境中非常重要的一個玩意兒，我們知道 Linux 是多人多工的環境，每個人登入系統都能取得一個 bash， 每個人都能夠使用 bash 下達 mail 這個指令來收受『自己』的郵件，問題是， bash 是如何得知你的郵件信箱是哪個檔案？ 這就需要『變數』的幫助啦！所以，你說變數重不重要呢？底下我們將介紹重要的環境變數、變數的取用與設定等資料， 呼呼！動動腦時間又來到囉！^_^

什麼是變數？

那麼，什麼是『變數』呢？簡單的說，就是讓某一個特定字串代表不固定的內容就是了。舉個大家在國中都會學到的數學例子， 那就是：『 y = ax + b 』這東西，在等號左邊的(y)就是變數，在等號右邊的(ax+b)就是變數內容。 要注意的是，左邊是未知數，右邊是已知數喔！ 講的更簡單一點，我們可以『用一個簡單的 "字眼" 來取代另一個比較複雜或者是容易變動的資料』。這有什麼好處啊？最大的好處就是『方便！』。

- **變數的可變性與方便性**

舉例來說，我們每個帳號的郵件信箱預設是以 MAIL 這個變數來進行存取的， 當 dmtsai 這個使用者登入時，他便會取得 MAIL 這個變數，而這個變數的內容其實就是 /var/spool/mail/dmtsai， 那如果 vbird 登入呢？他取得的 MAIL 這個變數的內容其實就是 /var/spool/mail/vbird 。 而我們使用信件讀取指令 mail 來讀取自己的郵件信箱時，嘿嘿，這支程式可以直接讀取 MAIL 這個變數的內容， 就能夠自動的分辨出屬於自己的信箱信件囉！這樣一來，設計程式的設計師就真的很方便的啦！

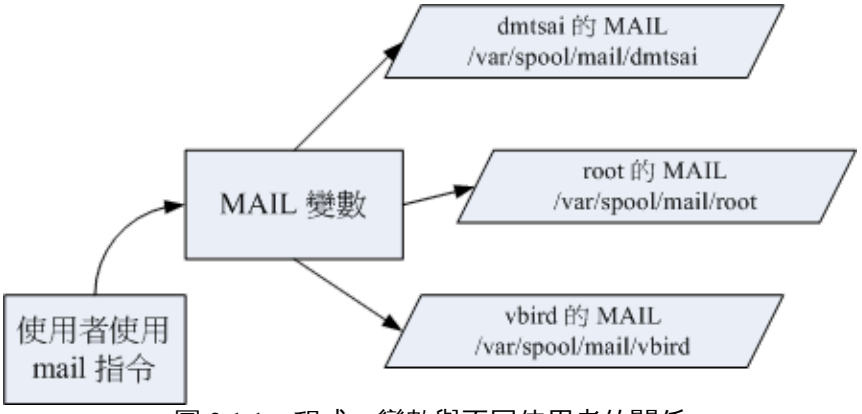

圖 2.1.1、程式、變數與不同使用者的關係

如上圖所示，由於系統已經幫我們規劃好 MAIL 這個變數，所以使用者只要知道 mail 這個指令如何使用即可， mail 會主動的取用 MAIL 這個變數，就能夠如上圖所示的取得自己的郵件信箱了！(注意大小寫，小寫的 mail 是指令， 大寫的 MAIL 則是變數名稱喔！)

那麼使用變數真的比較好嗎？這是當然的！想像一個例子，如果 mail 這個指令將 root 收信的郵件信箱 (mailbox) 檔名為 /var/spool/mail/root 直接寫入程式碼中。那麼當 dmtsai 要使用 mail 時，將會取得 /var/spool/mail/root 這個檔案的內容！ 不合理吧！所以你就需要幫 dmtsai 也設計一個 mail 的程式，將 /var/spool/mail/dmtsai 寫死到 mail 的程式碼當中！ 天吶！那系統要有多少個 mail 指令啊？反過來說，使用變數就變的很簡單了！因為你不需要更動到程式碼啊！ 只要將 MAIL 這個變數帶入不同的內容即可讓所有使用者透過 mail 取得自己的信件！當然簡單多了！

- **影響 bash 環境操作的變數**

某些特定變數會影響到 bash 的環境喔！舉例來說，我們前面已經提到過很多次的那個 PATH 變數！你能不能在任何目錄下執行某個指令，與 PATH 這個變數有很大的關係。例如你下達 ls 這個指令時，系統就是透過 PATH 這個變數裡面的內容所記錄的路徑順序來搜尋指令的呢！如果在搜尋完 PATH 變數內的路徑還找不到 ls 這個指令時， 就會在螢幕上顯示『 command not found 』的錯誤訊息了。

如果說的學理一點，那麼由於在 Linux System 下面，所有的執行緒都是需要一個執行碼， 而就如同上面提到的，你『真正以 shell 來跟 Linux 溝通，是在正確的登入 Linux 之後！』這個時候你就有一個 bash 的執行程序，也才可以真正的經由 bash 來跟系統溝通囉！而在進入 shell 之前，也正如同上面提到的，由於系統需要一些變數來提供他資料的存取 (或者是一些環境的設定參數值， 例如是否要顯示彩色等等的)，所以就有一些所謂的『環境變數』需要來讀入系統中了！這些環境變數例如 PATH、HOME、MAIL、SHELL 等等，都是很重要的， 為了區別與自訂變數的不同，環境變數通常以大寫字元來表示呢！

- **腳本程式設計（shell script）的好幫手**

這些還都只是系統預設的變數的目的，如果是個人的設定方面的應用呢：例如你要寫一個大型的 script 時，有些資料因為可能由於使用者習慣的不同而有差異，比如說路徑好了，由於該路徑在 script 被使用在相當多的地方，如果下次換了一部主機，都要修改 script 裡面的所有路徑，那麼我一定會瘋掉！ 這個時候如果使用變數，而將該變數的定義寫在最前面，後面相關的路徑名稱都以變數來取代，嘿嘿！那麼你只要修改一行就等於修改整篇 script 了！方便的很！所以，良好的程式設計師都會善用變數的定義！

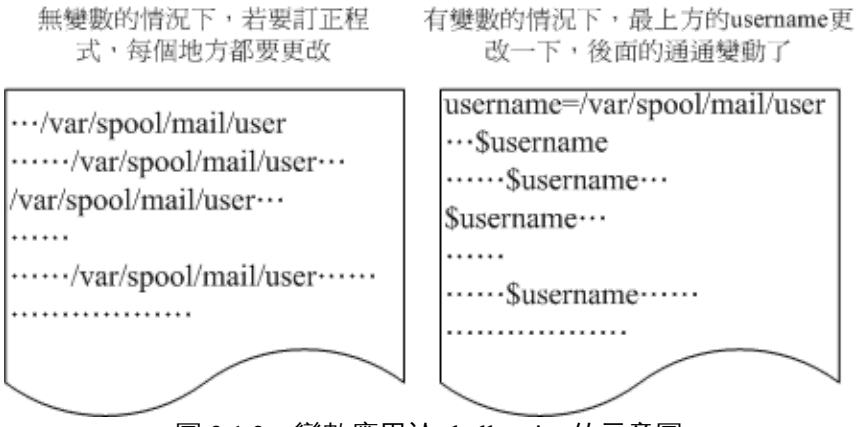

圖 2.1.2、變數應用於 shell script 的示意圖

最後我們就簡單的對『什麼是變數』作個簡單定義好了：『變數就是以一組文字或符號等，來取代一些設定或者是一串保留的資料！』， 例如：我設定了『myname』就是『VBird』，所以當你讀取 myname 這個變數的時候，系統自然就會知道！哈！那就是 VBird 啦！ 那麼如何『顯示變數』呢？這就需要使用到 echo 這個指令啦！

變數的取用與設定：echo，變數設定規則，unset

說的口沫橫飛的，也不知道『變數』與『變數代表的內容』有啥關係？ 那我們就將『變數』的『內容』拿出來給您瞧瞧好了。你可以利用 echo 這個指令來取用變數， 但是，變數在被取用時，前面必須要加上錢字號『 $ 』才行，舉例來說，要知道 PATH 的內容，該如何是好？

- **變數的取用：echo**

```
[root@www ~]# echo $variable
[root@www ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@www ~]# echo ${PATH}
```

變數的取用就如同上面的範例，利用 echo 就能夠讀出，只是需要在變數名稱前面加上 $， 或者是以 ${變數} 的方式來取用都可以！當然啦，那個 echo 的功能可是很多的， 我們這裡單純是拿 echo 來讀出變數的內容而已，更多的 echo 使用，請自行給他 man echo 吧！ ^_^

例題：

請在螢幕上面顯示出您的環境變數 HOME 與 MAIL：

答：

```
echo $HOME 或者是 echo ${HOME}
echo $MAIL 或者是 echo ${MAIL}
```

現在我們知道了變數與變數內容之間的相關性了，好了，那麼我要如何『設定』或者是『修改』某個變數的內容啊？很簡單啦！用『等號(=)』連接變數與他的內容就好啦！舉例來說： 我要將 myname 這個變數名稱的內容設定為 VBird， 那麼：

```
[root@www ~]# echo $myname
      <==這裡並沒有任何資料～因為這個變數尚未被設定！是空的！
[root@www ~]# myname=VBird
[root@www ~]# echo $myname
VBird  <==出現了！因為這個變數已經被設定了！
```

瞧！如此一來，這個變數名稱 myname 的內容就帶有 VBird 這個資料囉～ 而由上面的例子當中，我們也可以知道： 在 bash 當中，當一個變數名稱尚未被設定時，預設的內容是『空』的。 另外，變數在設定時，還是需要符合某些規定的，否則會設定失敗喔！ 這些規則如下所示啊！

- **變數的設定規則**

1. 變數與變數內容以一個等號『=』來連結，如下所示：
   『myname=VBird』

2. 等號兩邊不能直接接空白字元，如下所示為錯誤：
   『myname = VBird』或『myname=VBird Tsai』

3. 變數名稱只能是英文字母與數字，但是開頭字元不能是數字，如下為錯誤：
   『2myname=VBird』

4. 變數內容若有空白字元可使用雙引號『"』或單引號『'』將變數內容結合起來，但
   - 雙引號內的特殊字元如 $ 等，可以保有原本的特性，如下所示：
     『var="lang is $LANG"』則『echo $var』可得『lang is en_US』
   - 單引號內的特殊字元則僅為一般字元（純文字），如下所示：
     『var='lang is $LANG'』則『echo $var』可得『lang is $LANG』

5. 可用跳脫字元『 \ 』將特殊符號(如 [Enter], $, \, 空白字元, '等)變成一般字元；

6. 在一串指令中，還需要藉由其他的指令提供的資訊，可以使用反單引號『`指令`』或 『$(指令)』。特別注意，那個 ` 是鍵盤上方的數字鍵 1 左邊那個按鍵，而不是單引號！ 例如想要取得核心版本的設定：
   『version=$(uname -r)』再『echo $version』可得『2.6.18-128.el5』

7. 若該變數為擴增變數內容時，則可用 "$變數名稱" 或 ${變數} 累加內容，如下所示：
   『PATH="$PATH":/home/bin』

8. 若該變數需要在其他子程序執行，則需要以 export 來使變數變成環境變數：『export PATH』

9. 通常大寫字元為系統預設變數，自行設定變數可以使用小寫字元，方便判斷（純粹依照使用者興趣與嗜好）；

10. 取消變數的方法為使用 unset ：『unset 變數名稱』例如取消 myname 的設定：『unset myname』

底下讓鳥哥舉幾個例子來讓你試看看，就知道怎麼設定好你的變數囉！

```
範例一：設定一變數 name ，且內容為 VBird
[root@www ~]# 12name=VBird
-bash: 12name=VBird: command not found  <==螢幕會顯示錯誤！因為不能以數字開頭！
[root@www ~]# name = VBird            <==還是錯誤！因為有空白！
[root@www ~]# name=VBird              <==OK 的啦！

範例二：承上題，若變數內容為 VBird's name 呢，就是變數內容含有特殊符號時：
[root@www ~]# name=VBird's name
# 單引號與雙引號必須要成對，在上面的設定中僅有一個單引號，因此當你按下 enter 後，
# 你還可以繼續輸入變數內容。這與我們所需要的功能不同，失敗啦！
# 記得，失敗後要復原請按下 [ctrl]-c 結束！
[root@www ~]# name="VBird's name"     <==OK 的啦！
# 指令是由左邊向右找→，先遇到的引號先有用，因此如上所示，單引號會失效！
[root@www ~]# name='VBird's name'     <==失敗的啦！
# 因為前兩個單引號已成對，後面就多了一個不成對的單引號了！因此也就失敗了！
[root@www ~]# name=VBird\'s\ name     <==OK 的啦！
# 利用反斜線（\）跳脫特殊字元，例如單引號與空白鍵，這也是 OK 的啦！

範例三：我要在 PATH 這個變數當中『累加』:/home/dmtsai/bin 這個目錄
[root@www ~]# PATH=$PATH:/home/dmtsai/bin
[root@www ~]# PATH="$PATH":/home/dmtsai/bin
[root@www ~]# PATH=${PATH}:/home/dmtsai/bin
# 上面這三種格式在 PATH 裡頭的設定都是 OK 的！但是底下的例子就不見得囉！

範例四：承範例三，我要將 name 的內容多出 "yes" 呢？
[root@www ~]# name=$nameyes
# 知道了吧？如果沒有雙引號，那麼變數成了啥？name 的內容是 $nameyes 這個變數！
# 呵呵！我們可沒有設定過 nameyes 這個變數吶！所以，應該是底下這樣才對！
[root@www ~]# name="$name"yes
[root@www ~]# name=${name}yes  <==以此例較佳！

範例五：如何讓我剛剛設定的 name=VBird 可以用在下個 shell 的程序？
[root@www ~]# name=VBird
[root@www ~]# bash        <==進入到所謂的子程序
[root@www ~]# echo $name  <==子程序：再次的 echo 一下；
       <==嘿嘿！並沒有剛剛設定的內容喔！
[root@www ~]# exit        <==子程序：離開這個子程序
[root@www ~]# export name
[root@www ~]# bash        <==進入到所謂的子程序
[root@www ~]# echo $name  <==子程序：在此執行！
VBird  <==看吧！出現設定值了！
[root@www ~]# exit        <==子程序：離開這個子程序
```

什麼是『子程序』呢？就是說，在我目前這個 shell 的情況下，去啟用另一個新的 shell ，新的那個 shell 就是子程序啦！在一般的狀態下，父程序的自訂變數是無法在子程序內使用的。但是透過 export 將變數變成環境變數後，就能夠在子程序底下應用了！很不賴吧！至於程序的相關概念， 我們會在第

十七章程序管理當中提到的喔！

```
範例六：如何進入到您目前核心的模組目錄？
[root@www ~]# cd /lib/modules/`uname -r`/kernel
[root@www ~]# cd /lib/modules/$(uname -r)/kernel
```

每個 Linux 都能夠擁有多個核心版本，且幾乎 distribution 的核心版本都不相同。以 CentOS 5.3 (未更新前) 為例，他的預設核心版本是 2.6.18-128.el5 ， 所以核心模組目錄在 /lib/modules/2.6.18-128.el5/kernel/ 內。 也由於每個 distributions 的這個值都不相同，但是我們卻可以利用 uname -r 這個指令先取得版本資訊。所以囉，就可以透過上面指令當中的內含指令 `uname -r` 先取得版本輸出到 cd ... 那個指令當中，就能夠順利的進入目前核心的驅動程式所放置的目錄囉！很方便吧！

其實上面的指令可以說是作了兩次動作，亦即是：

1. 先進行反單引號內的動作『uname -r』並得到核心版本為 2.6.18-128.el5
2. 將上述的結果帶入原指令，故得指令為：『cd /lib/modules/2.6.18-128.el5/kernel/』

```
範例七：取消剛剛設定的 name 這個變數內容
[root@www ~]# unset name
```

根據上面的案例你可以試試看！就可以瞭解變數的設定囉！這個是很重要的呦！請勤加練習！ 其中，較為重要的一些特殊符號的使用囉！例如單引號、雙引號、跳脫字元、錢字號、反單引號等等，底下的例題想一想吧！

例題：

在變數的設定當中，單引號與雙引號的用途有何不同？

答：

單引號與雙引號的最大不同在於雙引號仍然可以保有變數的內容，但單引號內僅能是一般字元 ， 而不會有特殊符號。我們以底下的例子做說明：假設您定義了一個變數， name=VBird ， 現在想以 name 這個變數的內容定義出 myname 顯示 VBird its me 這個內容，要如何訂定呢？

```
[root@www ~]# name=VBird
[root@www ~]# echo $name
VBird
[root@www ~]# myname="$name its me"
[root@www ~]# echo $myname
VBird its me
[root@www ~]# myname='$name its me'
[root@www ~]# echo $myname
$name its me
```

發現了嗎？沒錯！使用了單引號的時候，那麼 $name 將失去原有的變數內容，僅為一般字元的顯示型態而已！這裡必需要特別小心在意！

例題：

在指令下達的過程中，反單引號(`)這個符號代表的意義為何？

答：

在一串指令中，在 ` 之內的指令將會被先執行，而其執行出來的結果將做為外部的輸入資訊！例如 uname -r 會顯示出目前的核心版本，而我們的核心版本在 /lib/modules 裡面，因此，你可以先執行 uname -r 找出核心版本，然後再以『 cd 目錄』到該目錄

下，當然也可以執行如同上面範例六的執行內容囉。

另外再舉個例子，我們也知道， locate 指令可以列出所有的相關檔案檔名，但是，如果我想要知道各個檔案的權限呢？舉例來說，我想要知道每個 crontab 相關檔名的權限：

```
[root@www ~]# ls -l `locate crontab`
```

如此一來，先以 locate 將檔名資料都列出來，再以 ls 指令來處理的意思啦！瞭了嗎？^_^

例題：

若你有一個常去的工作目錄名稱為：『/cluster/server/work/taiwan_2005/003/』，如何進行該目錄的簡化？

答：

在一般的情況下，如果你想要進入上述的目錄得要『cd /cluster/server/work/taiwan_2005/003/』， 以鳥哥自己的案例來說，鳥哥跑數值模式常常會設定很長的目錄名稱(避免忘記)，但如此一來變換目錄就很麻煩。 此時，鳥哥習慣利用底下的方式來降低指令下達錯誤的問題：

```
[root@www ~]# work="/cluster/server/work/taiwan_2005/003/"
[root@www ~]# cd $work
```

未來我想要使用其他目錄作為我的模式工作目錄時，只要變更 work 這個變數即可！而這個變數又可以在 bash 的設定檔中直接指定，那我每次登入只要執行『 cd $work 』就能夠去到數值模式模擬的工作目錄了！是否很方便呢？^_^

Tips:
老實說，使用『 version=$(uname -r) 』來取代『 version=`uname -r` 』比較好，因為反單引號大家老是容易打錯或看錯！ 所以現在鳥哥都習慣使用 $(指令) 來介紹這個功能！

環境變數的功能

環境變數可以幫我們達到很多功能～包括家目錄的變換啊、提示字元的顯示啊、執行檔搜尋的路徑啊等等的， 還有很多很多啦！那麼，既然環境變數有那麼多的功能，問一下，目前我的 shell 環境中，有多少預設的環境變數啊？我們可以利用兩個指令來查閱，分別是 env 與 export 呢！

用 env 觀察環境變數與常見環境變數說明

```
範例一：列出目前的 shell 環境下的所有環境變數與其內容。
[root@www ~]# env
HOSTNAME=www.vbird.tsai     <== 這部主機的主機名稱
TERM=xterm                  <== 這個終端機使用的環境是什麼類型
SHELL=/bin/bash             <== 目前這個環境下，使用的 Shell 是哪一個程式？
HISTSIZE=1000               <== 『記錄指令的筆數』在 CentOS 預設可記錄 1000 筆
USER=root                   <== 使用者的名稱啊！
LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd=40;33;01:
```

```
or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.btm=0
0;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=
00;31:*.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;3
1:*.tz=00;31:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00
;35:*.xpm=00;35:*.png=00;35:*.tif=00;35: <== 一些顏色顯示
MAIL=/var/spool/mail/root  <== 這個使用者所取用的 mailbox 位置
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr/local/sbin:
/root/bin                  <== 不再多講啊！是執行檔指令搜尋路徑
INPUTRC=/etc/inputrc       <== 與鍵盤按鍵功能有關。可以設定特殊按鍵！
PWD=/root                  <== 目前使用者所在的工作目錄（利用 pwd 取出！）
LANG=en_US                 <== 這個與語系有關，底下會再介紹！
HOME=/root                 <== 這個使用者的家目錄啊！
_=/bin/env                 <== 上一次使用的指令的最後一個參數(或指令本身)
```

env 是 environment (環境) 的簡寫啊，上面的例子當中，是列出來所有的環境變數。當然，如果使用 export 也會是一樣的內容～只不過， export 還有其他額外的功能就是了，我們等一下再提這個 export 指令。 那麼上面這些變數有些什麼功用呢？底下我們就一個一個來分析分析！

- **HOME**
  代表使用者的家目錄。還記得我們可以使用 cd ~ 去到自己的家目錄嗎？或者利用 cd 就可以直接回到使用者家目錄了。那就是取用這個變數啦～ 有很多程式都可能會取用到這個變數的值！

- **SHELL**
  告知我們，目前這個環境使用的 SHELL 是哪支程式？ Linux 預設使用 /bin/bash 的啦！

- **HISTSIZE**
  這個與『歷史命令』有關，亦即是， 我們曾經下達過的指令可以被系統記錄下來，而記錄的『筆數』則是由這個值來設定的。

- **MAIL**
  當我們使用 mail 這個指令在收信時，系統會去讀取的郵件信箱檔案 (mailbox)。

- **PATH**
  就是執行檔搜尋的路徑啦～目錄與目錄中間以冒號(:)分隔， 由於檔案的搜尋是依序由 PATH 的變數內的目錄來查詢，所以，目錄的順序也是重要的喔。

- **LANG**
  這個重要！就是語系資料囉～很多訊息都會用到他， 舉例來說，當我們在啟動某些 perl 的程式語言檔案時，他會主動的去分析語系資料檔案， 如果發現有他無法解析的編碼語系，可能會產生錯誤喔！一般來說，我們中文編碼通常是 zh_TW.Big5 或者是 zh_TW.UTF-8，這兩個編碼偏偏不容易被解譯出來，所以，有的時候，可能需要修訂一下語系資料。 這部分我們會在下個小節做介紹的！

- **RANDOM**
  這個玩意兒就是『隨機亂數』的變數啦！目前大多數的 distributions 都會有亂數產生器，那就是 /dev/random 這個檔案。 我們可以透過這個亂數檔案相關的變數 ($RANDOM) 來隨機取得亂數值喔。在 BASH 的環境下，這個 RANDOM 變數的內容，介於 0~32767 之間，所以，你只要 echo $RANDOM 時，系統就會主動的隨機取出一個介於 0~32767 的數值。萬一我想要使用 0~9 之間的數值呢？呵呵～利用 declare 宣告數值類型， 然後這樣做就可以了：

```
[root@www ~]# declare -i number=$RANDOM*10/32768 ; echo $number
8   <== 此時會隨機取出 0~9 之間的數值喔！
```

大致上是有這些環境變數啦～裡面有些比較重要的參數，在底下我們都會另外進行一些說明的～

用 set 觀察所有變數（含環境變數與自訂變數）

bash 可不只有環境變數喔，還有一些與 bash 操作介面有關的變數，以及使用者自己定義的變數存在的。 那麼這些變數如何觀察呢？這個時候就得要使用 set 這個指令了。 set 除了環境變數之外， 還會將其他在 bash 內的變數通通顯示出來哩！資訊很多，底下鳥哥僅列出幾個重要的內容：

```
[root@www ~]# set
BASH=/bin/bash            <== bash 的主程式放置路徑
BASH_VERSINFO=([0]="3" [1]="2" [2]="25" [3]="1" [4]="release"
[5]="i686-redhat-linux-gnu")      <== bash 的版本啊！
BASH_VERSION='3.2.25(1)-release'  <== 也是 bash 的版本啊！
COLORS=/etc/DIR_COLORS.xterm      <== 使用的顏色紀錄檔案
COLUMNS=115               <== 在目前的終端機環境下，使用的欄位有幾個字元長度
HISTFILE=/root/.bash_history      <== 歷史命令記錄的放置檔案，隱藏檔
HISTFILESIZE=1000         <== 存起來(與上個變數有關)的檔案之指令的最大紀錄筆數。
HISTSIZE=1000             <== 目前環境下，可記錄的歷史命令最大筆數。
HOSTTYPE=i686             <== 主機安裝的軟體主要類型。我們用的是 i686 相容機器軟體
IFS=$' \t\n'              <== 預設的分隔符號
LINES=35                  <== 目前的終端機下的最大行數
MACHTYPE=i686-redhat-linux-gnu    <== 安裝的機器類型
MAILCHECK=60              <== 與郵件有關。每 60 秒去掃瞄一次信箱有無新信！
OLDPWD=/home              <== 上個工作目錄。我們可以用 cd - 來取用這個變數。
OSTYPE=linux-gnu          <== 作業系統的類型！
PPID=20025                <== 父程序的 PID（會在後續章節才介紹）
PS1='[\u@\h \W]\$ '       <== PS1 就厲害了。這個是命令提示字元，也就是我們常見的
                              [root@www ~]# 或 [dmtsai ~]$ 的設定值啦！可以更動的！
PS2='> '                  <== 如果你使用跳脫符號（\）第二行以後的提示字元也
name=VBird                <== 剛剛設定的自訂變數也可以被列出來喔！
$                         <== 目前這個 shell 所使用的 PID
?                         <== 剛剛執行完指令的回傳值。
```

一般來說，不論是否為環境變數，只要跟我們目前這個 shell 的操作介面有關的變數， 通常都會被設定為大寫字元，也就是說，『基本上，在 Linux 預設的情況中，使用{大寫的字母}來設定的變數一般為系統內定需要的變數』。 OK！OK！那麼上頭那些變數當中，有哪些是比較重要的？大概有這幾個吧！

- **PS1**：(提示字元的設定)

  這是 PS1 (數字的 1 不是英文字母)，這個東西就是我們的『命令提示字元』喔！ 當我們每次按下 [Enter] 按鍵去執行某個指令後，最後要再次出現提示字元時， 就會主動去讀取這個變數值了。上頭 PS1 內顯示的是一些特殊符號，這些特殊符號可以顯示不同的資訊， 每個 distributions 的 bash 預設的 PS1 變數內容可能有些許的差異，不要緊，『習慣你自己的習慣』就好了。 你可以用 man bash (註3)去查詢一下 PS1 的相關說明，以理解底下的一些符號意義。

  - \d ：可顯示出『星期 月 日』的日期格式，如："Mon Feb 2"
  - \H ：完整的主機名稱。舉例來說，鳥哥的練習機為『www.vbird.tsai』
  - \h ：僅取主機名稱在第一個小數點之前的名字，如鳥哥主機則為『www』後面省略
  - \t ：顯示時間，為 24 小時格式的『HH:MM:SS』
  - \T ：顯示時間，為 12 小時格式的『HH:MM:SS』
  - \A ：顯示時間，為 24 小時格式的『HH:MM』
  - \@ ：顯示時間，為 12 小時格式的『am/pm』樣式
  - \u ：目前使用者的帳號名稱，如『root』；
  - \v ：BASH 的版本資訊，如鳥哥的測試主機版本為 3.2.25(1)，僅取『3.2』顯示
  - \w ：完整的工作目錄名稱，由根目錄寫起的目錄名稱。但家目錄會以 ~ 取代；
  - \W ：利用 basename 函數取得工作目錄名稱，所以僅會列出最後一個目錄名。
  - \# ：下達的第幾個指令。
  - \$ ：提示字元，如果是 root 時，提示字元為 # ，否則就是 $ 囉～

好了，讓我們來看看 CentOS 預設的 PS1 內容吧：『[\u@\h \W]\$ 』，現在你知道那些反斜線後的資料意義了吧？ 要注意喔！那個反斜線後的資料為 PS1 的特殊功能，與 bash 的變數設定沒關係啦！不要搞混了喔！ 那你現在知道為何你的命令提示字元是：『 [root@www ~]# 』了吧？ 好了，那麼假設我想要有類似底下的提示字元：

```
[root@www /home/dmtsai 16:50 #12]#
```

那個 # 代表第 12 次下達的指令。那麼應該如何設定 PS1 呢？可以這樣啊：

```
[root@www ~ ]# cd /home
[root@www home]# PS1='[\u@\h \w \A #\#]\$ '
[root@www /home 17:02 #85]#
# 看到了嗎？提示字元變了！變的很有趣吧！其中，那個 #85 比較有趣，
# 如果您再隨便輸入幾次 ls 後，該數字就會增加喔！為啥？上面有說明滴！
```

- **$**：(關於本 shell 的 PID)

  錢字號本身也是個變數喔！這個咚咚代表的是『目前這個 Shell 的執行緒代號』，亦即是所謂的 PID (Process ID)。 更多的程序觀念，我們會在第四篇的時候提及。想要知道我們的 shell 的 PID，就可以用：『 echo $$ 』即可！出現的數字就是你的 PID 號碼。

- **?**：(關於上個執行指令的回傳值)

  蝦密？問號也是一個特殊的變數？沒錯！在 bash 裡面這個變數可重要的很！ 這個變數是：『上一個執行的指令所回傳的值』， 上面這句話的重點是『上一個指令』與『回傳值』兩個地方。當我們執行某些指令時， 這些指令都會回傳一個執行後的代碼。一般來說，如果成功的執行該指令，則會回傳一個 0 值，如果執行過程發生錯誤，就會回傳『錯誤代碼』才對！一般就是以非為 0 的數值來取代。 我們以底下的例子來看看：

```
[root@www ~]# echo $SHELL
/bin/bash                                   <==可順利顯示！沒有錯誤！
[root@www ~]# echo $?
0                                           <==因為沒問題，所以回傳值為 0
[root@www ~]# 12name=VBird
-bash: 12name=VBird: command not found      <==發生錯誤了！bash回報有問題
[root@www ~]# echo $?
127                                         <==因為有問題，回傳錯誤代碼(非為0)
# 錯誤代碼回傳值依據軟體而有不同，我們可以利用這個代碼來搜尋錯誤的原因喔！
[root@www ~]# echo $?
0
# 咦！怎麼又變成正確了？這是因為 "?" 只與『上一個執行指令』有關，
# 所以，我們上一個指令是執行『 echo $? 』，當然沒有錯誤，所以是 0 沒錯！
```

- **OSTYPE, HOSTTYPE, MACHTYPE**：(主機硬體與核心的等級)

  我們在第零章、計算機概論內的 CPU 等級說明中談過 CPU， 目前個人電腦的 CPU 主要分為 32/64 位元，其中 32 位元又可分為 i386, i586, i686，而 64 位元則稱為 x86_64。 由於不同等級的 CPU 指令集不太相同，因此你的軟體可能會針對某些 CPU 進行最佳化，以求取較佳的軟體性能。所以軟體就有 i386, i686 及 x86_64 之分。以目前 (2009) 的主流硬體來說，幾乎都是 x86_64 的天下！ 但是畢竟舊機器還是非常多，以鳥哥的環境來說，我用 P-III 等級的電腦，所以上頭就發現我的等級是 i686 啦！

  要留意的是，較高階的硬體通常會向下相容舊有的軟體，但較高階的軟體可能無法在舊機器上面安裝！ 我們在第三章就曾說明過， 這裡再強調一次，你可以在 x86_64 的硬體上安裝 i386 的 Linux

作業系統，但是你無法在 i686 的硬體上安裝 x86_64 的 Linux 作業系統！這點得要牢記在心！

- **export： 自訂變數轉成環境變數**

談了 env 與 set 現在知道有所謂的環境變數與自訂變數，那麼這兩者之間有啥差異呢？其實這兩者的差異在於『該變數是否會被子程序所繼續引用』啦！唔！那麼啥是父程序？子程序？ 這就得要瞭解一下指令的下達行為了。

當你登入 Linux 並取得一個 bash 之後，你的 bash 就是一個獨立的程序，被稱為 PID 的就是。 接下來你在這個 bash 底下所下達的任何指令都是由這個 bash 所衍生出來的，那些被下達的指令就被稱為子程序了。 我們可以用底下的圖示來簡單的說明一下父程序與子程序的概念：

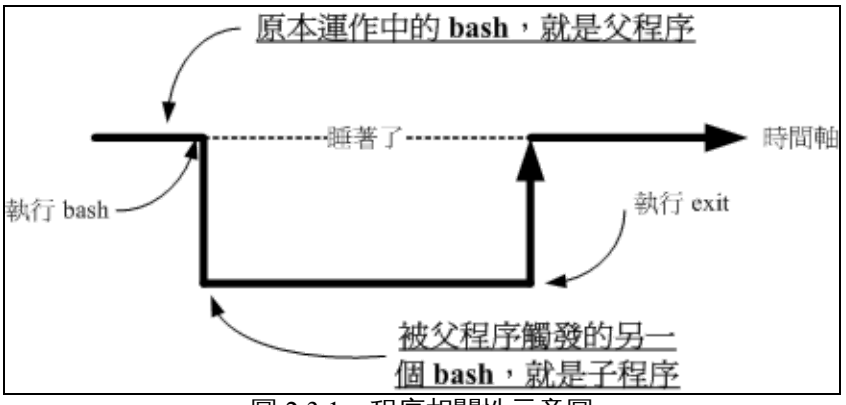

圖 2.3.1、程序相關性示意圖

如上所示，我們在原本的 bash 底下執行另一個 bash， 結果操作的環境介面會跑到第二個 bash 去(就是子程序)， 那原本的 bash 就會在暫停的情況 (睡著了， 就是 sleep)。整個指令運作的環境是實線的部分！若要回到原本的 bash 去， 就只有將第二個 bash 結束掉 (下達 exit 或 logout) 才行。更多的程序概念我們會在第四篇談及，這裡只要有這個概念即可。

這個程序概念與變數有啥關係啊？關係可大了！因為子程序僅會繼承父程序的環境變數， 子程序不會繼承父程序的自訂變數啦！所以你在原本 bash 的自訂變數在進入了子程序後就會消失不見， 一直到你離開子程序並回到原本的父程序後，這個變數才會又出現！

換個角度來想，也就是說，如果我能將自訂變數變成環境變數的話，那不就可以讓該變數值繼續存在於子程序了？ 呵呵！沒錯！此時，那個 export 指令就很有用啦！如你想要讓該變數內容繼續的在子程序中使用，那麼就請執行：

```
[root@www ~]# export 變數名稱
```

這東西用在『分享自己的變數設定給後來呼叫的檔案或其他程序』啦！ 像鳥哥常常在自己的主檔案後面呼叫其他附屬檔案(類似函式的功能)，但是主檔案與附屬檔案內都有相同的變數名稱， 若一再重複設定時，要修改也很麻煩，此時只要在原本的第一個檔案內設定好『 export 變數 』， 後面所呼叫的檔案就能夠使用這個變數設定了！而不需要重複設定，這非常實用於 shell script 當中喔！ 如果僅下達 export 而沒有接變數時，那麼此時將會把所有的『環境變數』秀出來喔！例如：

```
[root@www ~]# export
declare -x HISTSIZE="1000"
declare -x HOME="/root"
declare -x HOSTNAME="www.vbird.tsai"
declare -x INPUTRC="/etc/inputrc"
declare -x LANG="en_US"
declare -x LOGNAME="root"
# 後面的鳥哥就都直接省略了！不然....浪費版面～ ^_^
```

那如何將環境變數轉成自訂變數呢？可以使用本章後續介紹的 declare 呢！

影響顯示結果的語系變數（locale）

還記得我們在第五章裡面提到的語系問題嗎？ 就是當我們使用 man command 的方式去查詢某個資料的說明檔時，該說明檔的內容可能會因為我們使用的語系不同而產生亂碼。 另外，利用 ls 查詢檔案的時間時，也可能會有亂碼出現在時間的部分。那個問題其實就是語系的問題啦。

目前大多數的 Linux distributions 已經都是支援日漸流行的萬國碼了，也都支援大部分的國家語系。 這有賴於 i18n (註4) 支援的幫助呢！ 那麼我們的 Linux 到底支援了多少的語系呢？這可以由 locale 這個指令來查詢到喔！

```
[root@www ~]# locale -a
....(前面省略)....
zh_TW
zh_TW.big5      <==大五碼的中文編碼
zh_TW.euctw
zh_TW.utf8      <==萬國碼的中文編碼
zu_ZA
zu_ZA.iso88591
zu_ZA.utf8
```

正體中文語系至少支援了兩種以上的編碼，一種是目前還是很常見的 big5， 另一種則是越來越熱門的 utf-8 編碼。 那麼我們如何修訂這些編碼呢？其實可以透過底下這些變數的說：

```
[root@www ~]# locale  <==後面不加任何選項與參數即可！
LANG=en_US                  <==主語言的環境
LC_CTYPE="en_US"            <==字元(文字)辨識的編碼
LC_NUMERIC="en_US"          <==數字系統的顯示訊息
LC_TIME="en_US"             <==時間系統的顯示資料
LC_COLLATE="en_US"          <==字串的比較與排序等
LC_MONETARY="en_US"         <==幣值格式的顯示等
LC_MESSAGES="en_US"         <==訊息顯示的內容，如功能表、錯誤訊息等
LC_ALL=                     <==整體語系的環境
....(後面省略)....
```

基本上，你可以逐一設定每個與語系有關的變數資料，但事實上，如果其他的語系變數都未設定， 且你有設定 LANG 或者是 LC_ALL 時，則其他的語系變數就會被這兩個變數所取代！ 這也是為什麼我們在 Linux 當中，通常說明僅設定 LANG 這個變數而已，因為他是最主要的設定變數！ 好了，那麼你應該要覺得奇怪的是，為什麼在 Linux 主機的終端機介面 (tty1 ~ tty6) 的環境下，如果設定『 LANG=zh_TW.big5 』這個設定值生效後，使用 man 或者其他訊息輸出時， 都會有一堆亂碼，尤其是使用 ls -l 這個參數時？

因為在 Linux 主機的終端機介面環境下是無法顯示像中文這麼複雜的編碼文字， 所以就會產生亂碼了。也就是如此，我們才會必須要在 tty1 ~ tty6 的環境下， 加裝一些中文化介面的軟體，才能夠看到中文啊！不過，如果你是在 MS Windows 主機以遠端連線伺服器的軟體連線到主機的話，那麼，嘿嘿！其實文字介面確實是可以看到中文的。 此時反而你得要在 LANG 設定中文編碼才好呢！

> **Tips:**
> 無論如何，如果發生一些亂碼的問題，那麼設定系統裡面保有的語系編碼， 例如： en_US 或 en_US.utf8 等等的設定，應該就 OK 的啦！好了，那麼系統預設支援多少種語系呢？ 當我們使用 locale 時，系統是列出目前 Linux 主機內保有的語系檔案， 這些語系檔案都放置在：

/usr/lib/locale/ 這個目錄中。

你當然可以讓每個使用者自己去調整自己喜好的語系，但是整體系統預設的語系定義在哪裡呢？ 其實就是在 /etc/sysconfig/i18n 囉！這個檔案在 CentOS 5.x 的內容有點像這樣：

```
[root@www ~]# cat /etc/sysconfig/i18n
LANG="zh_TW.UTF-8"
```

因為鳥哥在第四章的安裝時選擇的是中文語系安裝畫面， 所以這個檔案預設就會使用中文編碼啦！你也可以自行將他改成你想要的語系編碼即可。

Tips:
假設你有一個純文字檔案原本是在 Windows 底下建立的，那麼這個檔案預設應該是 big5 的編碼格式。 在你將這個檔案上傳到 Linux 主機後，在 X window 底下打開時，咦！怎麼中文字通通變成亂碼了？ 別擔心！因為如上所示， i18n 預設是萬國碼系統嘛！你只要將開啟該檔案的軟體編碼由 utf8 改成 big5 就能夠看到正確的中文了！

變數的有效範圍

蝦密？變數也有使用的『範圍』？沒錯啊～我們在上頭的 export 指令說明中，就提到了這個概念了。如果在跑程式的時候，有父程序與子程序的不同程序關係時， 則『變數』可否被引用與 export 有關。被 export 後的變數，我們可以稱他為『環境變數』！ 環境變數可以被子程序所引用，但是其他的自訂變數內容就不會存在於子程序中。

Tips:
在某些不同的書籍會談到『全域變數, global variable』與『區域變數, local variable』。 基本上你可以這樣看待：
環境變數=全域變數
自訂變數=區域變數

在學理方面，為什麼環境變數的資料可以被子程序所引用呢？這是因為記憶體配置的關係！理論上是這樣的：

- 當啟動一個 shell，作業系統會分配一記憶區塊給 shell 使用，此記憶體內之變數可讓子程序取用
- 若在父程序利用 export 功能，可以讓自訂變數的內容寫到上述的記憶區塊當中(環境變數)；
- 當載入另一個 shell 時 (亦即啟動子程序，而離開原本的父程序了)，子 shell 可以將父 shell 的環境變數所在的記憶區塊導入自己的環境變數區塊當中。

透過這樣的關係，我們就可以讓某些變數在相關的程序之間存在，以幫助自己更方便的操作環境喔！不過要提醒的是，這個『環境變數』與『bash 的操作環境』意思不太一樣，舉例來說， PS1 並不是環境變數， 但是這個 PS1 會影響到 bash 的介面 (提示字元嘛)！相關性要釐清喔！^_^

變數鍵盤讀取、陣列與宣告： read, array, declare

我們上面提到的變數設定功能，都是由指令列直接設定的，那麼，可不可以讓使用者能夠經由鍵盤輸入？什麼意思呢？是否記得某些程式執行的過程當中，會等待使用者輸入 "yes/no" 之類的訊息啊？在 bash 裡面也有相對應的功能喔！此外，我們還可以宣告這個變數的屬性， 例如：陣列或者是數字等等的。底下就來看看吧！

- **read**

要讀取來自鍵盤輸入的變數，就是用 read 這個指令了。這個指令最常被用在 shell script 的撰寫當中，想要跟使用者對談？用這個指令就對了。關於 script 的寫法，我們會在第十三章介紹，底下先來瞧一瞧 read 的相關語法吧！

```
[root@www ~]# read [-pt] variable
選項與參數：
-p   ：後面可以接提示字元！
-t   ：後面可以接等待的『秒數！』這個比較有趣～不會一直等待使用者啦！

範例一：讓使用者由鍵盤輸入一內容，將該內容變成名為 atest 的變數
[root@www ~]# read atest
This is a test        <==此時游標會等待你輸入！請輸入左側文字看看
[root@www ~]# echo $atest
This is a test          <==你剛剛輸入的資料已經變成一個變數內容！

範例二：提示使用者 30 秒內輸入自己的大名，將該輸入字串作為名為 named 的變數內容
[root@www ~]# read -p "Please keyin your name: " -t 30 named
Please keyin your name: VBird Tsai   <==注意看，會有提示字元喔！
[root@www ~]# echo $named
VBird Tsai        <==輸入的資料又變成一個變數的內容了！
```

read 之後不加任何參數，直接加上變數名稱，那麼底下就會主動出現一個空白行等待你的輸入(如範例一)。 如果加上 -t 後面接秒數，例如上面的範例二，那麼 30 秒之內沒有任何動作時， 該指令就會自動略過了～如果是加上 -p，嘿嘿！在輸入的游標前就會有比較多可以用的提示字元給我們參考！ 在指令的下達裡面，比較美觀啦！ ^_^

- **declare / typeset**

declare 或 typeset 是一樣的功能，就是在『宣告變數的類型』。如果使用 declare 後面並沒有接任何參數，那麼 bash 就會主動的將所有的變數名稱與內容通通叫出來，就好像使用 set 一樣啦！ 那麼 declare 還有什麼語法呢？看看先：

```
[root@www ~]# declare [-aixr] variable
選項與參數：
-a   ：將後面名為 variable 的變數定義成為陣列 (array) 類型
-i   ：將後面名為 variable 的變數定義成為整數數字 (integer) 類型
-x   ：用法與 export 一樣，就是將後面的 variable 變成環境變數；
-r   ：將變數設定成為 readonly 類型，該變數不可被更改內容，也不能 unset

範例一：讓變數 sum 進行 100+300+50 的加總結果
[root@www ~]# sum=100+300+50
[root@www ~]# echo $sum
100+300+50  <==咦！怎麼沒有幫我計算加總？因為這是文字型態的變數屬性啊！
[root@www ~]# declare -i sum=100+300+50
[root@www ~]# echo $sum
```

```
450          <==瞭乎？？
```

由於在預設的情況底下， bash 對於變數有幾個基本的定義：

- 變數類型預設為『字串』，所以若不指定變數類型，則 1+2 為一個『字串』而不是『計算式』。 所以上述第一個執行的結果才會出現那個情況的；
- bash 環境中的數值運算，預設最多僅能到達整數形態，所以 1/3 結果是 0；

現在你曉得為啥你需要進行變數宣告了吧？如果需要非字串類型的變數，那就得要進行變數的宣告才行啦！ 底下繼續來玩些其他的 declare 功能。

```
範例二：將 sum 變成環境變數
[root@www ~]# declare -x sum
[root@www ~]# export | grep sum
declare -ix sum="450"  <==果然出現了！包括有 i 與 x 的宣告！

範例三：讓 sum 變成唯讀屬性，不可更動！
[root@www ~]# declare -r sum
[root@www ~]# sum=tesgting
-bash: sum: readonly variable  <==老天爺～不能改這個變數了！

範例四：讓 sum 變成非環境變數的自訂變數吧！
[root@www ~]# declare +x sum  <== 將 - 變成 + 可以進行『取消』動作
[root@www ~]# declare -p sum  <== -p 可以單獨列出變數的類型
declare -ir sum="450" <== 看吧！只剩下 i, r 的類型，不具有 x 囉！
```

declare 也是個很有用的功能～尤其是當我們需要使用到底下的陣列功能時， 他也可以幫我們宣告陣列的屬性喔！不過，老話一句，陣列也是在 shell script 比較常用的啦！ 比較有趣的是，如果你不小心將變數設定為『唯讀』，通常得要登出再登入才能復原該變數的類型了！ @_@

- **陣列（array）變數類型**

某些時候，我們必須使用陣列來宣告一些變數，這有什麼好處啊？在一般人的使用上， 果然是看不出來有什麼好處的！不過，如果您曾經寫過程式的話，那才會比較瞭解陣列的意義～ 陣列對寫數值程式的設計師來說，可是不能錯過學習的重點之一哩！好！不囉唆～ 那麼要如何設定陣列的變數與內容呢？在 bash 裡頭，陣列的設定方式是：

```
var[index]=content
```

意思是說，我有一個陣列名稱為 var ， 而這個陣列的內容為 var[1]=小明， var[2]=大明， var[3]=好明 .... 等等，那個 index 就是一些數字啦，重點是用中刮號 ([ ]) 來設定的。 目前我們 bash 提供的是一維陣列。老實說，如果您不必寫一些複雜的程式， 那麼這個陣列的地方，可以先略過，等到有需要再來學習即可！因為要製作出陣列， 通常與迴圈或者其他判斷式交互使用才有比較高的存在意義！

```
範例：設定上面提到的 var[1] ～ var[3] 的變數。
[root@www ~]# var[1]="small min"
[root@www ~]# var[2]="big min"
[root@www ~]# var[3]="nice min"
[root@www ~]# echo "${var[1]}, ${var[2]}, ${var[3]}"
small min, big min, nice min
```

陣列的變數類型比較有趣的地方在於『讀取』，一般來說，建議直接以 ${陣列} 的方式來讀取，比較正確無誤的啦！

與檔案系統及程序的限制關係： ulimit

想像一個狀況：我的 Linux 主機裡面同時登入了十個人，這十個人不知怎麼搞的， 同時開啟了 100 個檔案，每個檔案的大小約 10MBytes，請問一下， 我的 Linux 主機的記憶體要有多大才夠？ 10*100*10 = 10000 MBytes = 10GBytes ... 老天爺，這樣，系統不掛點才有鬼哩！為了要預防這個情況的發生，所以我們的 bash 是可以『限制使用者的某些系統資源』的，包括可以開啟的檔案數量，可以使用的 CPU 時間，可以使用的記憶體總量等等。如何設定？用 ulimit 吧！

```
[root@www ~]# ulimit [-SHacdfltu] [配額]
選項與參數：
-H  ：hard limit ，嚴格的設定，必定不能超過這個設定的數值；
-S  ：soft limit ，警告的設定，可以超過這個設定值，但是若超過則有警告訊息。
      在設定上，通常 soft 會比 hard 小，舉例來說，soft 可設定為 80 而 hard
      設定為 100，那麼你可以使用到 90（因為沒有超過 100），但介於 80~100 之間時，
      系統會有警告訊息通知你！
-a  ：後面不接任何選項與參數，可列出所有的限制額度；
-c  ：當某些程式發生錯誤時，系統可能會將該程式在記憶體中的資訊寫成檔案(除錯用)，
      這種檔案就被稱為核心檔案(core file)。此為限制每個核心檔案的最大容量。
-f  ：此 shell 可以建立的最大檔案容量(一般可能設定為 2GB)單位為 Kbytes
-d  ：程序可使用的最大斷裂記憶體(segment)容量；
-l  ：可用於鎖定 (lock) 的記憶體量
-t  ：可使用的最大 CPU 時間 (單位為秒)
-u  ：單一使用者可以使用的最大程序(process)數量。

範例一：列出你目前身份(假設為root)的所有限制資料數值
[root@www ~]# ulimit -a
core file size          (blocks, -c) 0          <==只要是 0 就代表沒限制
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited  <==可建立的單一檔案的大小
pending signals                 (-i) 11774
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024       <==同時可開啟的檔案數量
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 10240
cpu time               (seconds, -t) unlimited
max user processes              (-u) 11774
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited

範例二：限制使用者僅能建立 10MBytes 以下的容量的檔案
[root@www ~]# ulimit -f 10240
[root@www ~]# ulimit -a
file size               (blocks, -f) 10240 <==最大量為10240Kbyes，相當10Mbytes
[root@www ~]# dd if=/dev/zero of=123 bs=1M count=20
File size limit exceeded <==嘗試建立 20MB 的檔案，結果失敗了！
```

還記得我們在第八章 Linux 磁碟檔案系統裡面提到過，單一 filesystem 能夠支援的單一檔案大小與 block 的大小有關。例如 block size 為 1024 byte 時，單一檔案可達 16GB 的容量。但是，我們可以用 ulimit 來限制使用者可以建立的檔案大小喔！ 利用 ulimit -f 就可以來設定了！例如上面的範例二，要注意單位喔！單位是 Kbytes。 若改天你一直無法建立一個大容量的檔案，記得瞧一瞧 ulimit 的資訊喔！

Tips:
想要復原 ulimit 的設定最簡單的方法就是登出再登入，否則就是得要重新以 ulimit 設定才行！ 不過，要注意的是，一般身份使用者如果以 ulimit 設定了 -f 的檔案大小， 那麼他『只能繼續減小檔案容量，不能增加檔案容量喔！』另外，若想要管控使用者的 ulimit 限值， 可以參考第十四章的 pam 的介紹。

變數內容的刪除、取代與替換

變數除了可以直接設定來修改原本的內容之外，有沒有辦法透過簡單的動作來將變數的內容進行微調呢？ 舉例來說，進行變數內容的刪除、取代與替換等！是可以的！我們可以透過幾個簡單的小步驟來進行變數內容的微調喔！ 底下就來試試看！

變數內容的刪除與取代

變數的內容可以很簡單的透過幾個咚咚來進行刪除喔！我們使用 PATH 這個變數的內容來做測試好了。 請你依序進行底下的幾個例子來玩玩，比較容易感受的到鳥哥在這裡想要表達的意義：

```
範例一：先讓小寫的 path 自訂變數設定的與 PATH 內容相同
[root@www ~]# path=${PATH}
[root@www ~]# echo $path
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！

範例二：假設我不喜歡 kerberos，所以要將前兩個目錄刪除掉，如何顯示？
[root@www ~]# echo ${path#/*kerberos/bin:}
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

上面這個範例很有趣的！他的重點可以用底下這張表格來說明：

```
${variable#/*kerberos/bin:}
   上面的特殊字體部分是關鍵字！用在這種刪除模式所必須存在的

${variable#/*kerberos/bin:}
   這就是原本的變數名稱，以上面範例二來說，這裡就填寫 path 這個『變數名稱』啦！

${variable#/*kerberos/bin:}
   這是重點！代表『從變數內容的最前面開始向右刪除』，且僅刪除最短的那個

${variable#/*kerberos/bin:}
   代表要被刪除的部分，由於 # 代表由前面開始刪除，所以這裡便由開始的 / 寫起。
   需要注意的是，我們還可以透過萬用字元 * 來取代 0 到無窮多個任意字元

   以上面範例二的結果來看， path 這個變數被刪除的內容如下所示：
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！
```

很有趣吧！這樣瞭解了 # 的功能了嗎？接下來讓我們來看看底下的範例三！

```
範例三：我想要刪除前面所有的目錄，僅保留最後一個目錄
[root@www ~]# echo ${path#/*:}
```

```
/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
/root/bin      <==這兩行其實是同一行啦！
# 由於一個 # 僅刪除掉最短的那個，因此他刪除的情況可以用底下的刪除線來看：
# /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
# /usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！

[root@www ~]# echo ${path##/*:}
/root/bin
# 嘿！多加了一個 # 變成 ## 之後，他變成『刪除掉最長的那個資料』！亦即是：
# /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
# /usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！
```

非常有趣！不是嗎？因為在 PATH 這個變數的內容中，每個目錄都是以冒號『:』隔開的， 所以要從頭刪除掉目錄就是介於斜線 (/) 到冒號 (:) 之間的資料！但是 PATH 中不止一個冒號 (:) 啊！ 所以 # 與 ## 就分別代表：

- # ：符合取代文字的『最短的』那一個；
- ##：符合取代文字的『最長的』那一個

上面談到的是『從前面開始刪除變數內容』，那麼如果想要『從後面向前刪除變數內容』呢？ 這個時候就得使用百分比 (%) 符號了！來看看範例四怎麼做吧！

```
範例四：我想要刪除最後面那個目錄，亦即從 : 到 bin 為止的字串
[root@www ~]# echo ${path%:*bin}
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin  <==注意啊！最後面一個目錄不見去！
# 這個 % 符號代表由最後面開始向前刪除！所以上面得到的結果其實是來自如下：
# /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
# /usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！

範例五：那如果我只想要保留第一個目錄呢？
[root@www ~]# echo ${path%%:*bin}
/usr/kerberos/sbin
# 同樣的， %% 代表的則是最長的符合字串，所以結果其實是來自如下：
# /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
# /usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！
```

由於我是想要由變數內容的後面向前面刪除，而我這個變數內容最後面的結尾是『/root/bin』， 所以你可以看到上面我刪除的資料最終一定是『bin』，亦即是『:*bin』那個 * 代表萬用字元！ 至於 % 與 %% 的意義其實與 # 及 ## 類似！這樣理解否？

例題：

假設你是 root ， 那你的 MAIL 變數應該是 /var/spool/mail/root 。假設你只想要保留最後面那個檔名 (root)， 前面的目錄名稱都不要了，如何利用 $MAIL 變數來達成？

答：

題意其實是這樣『/var/spool/mail/root』，亦即刪除掉兩條斜線間的所有資料(最長符合)。這個時候你就可以這樣做即可：

```
[root@www ~]# echo ${MAIL##/*/}
```

相反的，如果你只想要拿掉檔名，保留目錄的名稱，亦即是『/var/spool/mail/root』 (最短符合)。但假設你並不知道結尾的字母為何，此時你可以利用萬用字元來處理即可，如下所示：

```
[root@www ~]# echo ${MAIL%/*}
```

瞭解了刪除功能後，接下來談談取代吧！繼續玩玩範例六囉！

```
範例六：將 path 的變數內容內的 sbin 取代成大寫 SBIN：
[root@www ~]# echo ${path/sbin/SBIN}
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin
# 這個部分就容易理解的多了！關鍵字在於那兩個斜線，兩斜線中間的是舊字串
# 後面的是新字串，所以結果就會出現如上述的特殊字體部分囉！

[root@www ~]# echo ${path//sbin/SBIN}
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/SBIN:/usr/local/bin:/SBIN:/bin:
/usr/SBIN:/usr/bin:/root/bin
# 如果是兩條斜線，那麼就變成所有符合的內容都會被取代喔！
```

我們將這部份作個總結說明一下：

變數設定方式	說明
${變數#關鍵字} ${變數##關鍵字}	若變數內容從頭開始的資料符合『關鍵字』，則將符合的最短資料刪除 若變數內容從頭開始的資料符合『關鍵字』，則將符合的最長資料刪除
${變數%關鍵字} ${變數%%關鍵字}	若變數內容從尾向前的資料符合『關鍵字』，則將符合的最短資料刪除 若變數內容從尾向前的資料符合『關鍵字』，則將符合的最長資料刪除
${變數/舊字串/新字串} ${變數//舊字串/新字串}	若變數內容符合『舊字串』則『第一個舊字串會被新字串取代』 若變數內容符合『舊字串』則『全部的舊字串會被新字串取代』

- **變數的測試與內容替換**

在某些時刻我們常常需要『判斷』某個變數是否存在，若變數存在則使用既有的設定，若變數不存在則給予一個常用的設定。我們舉底下的例子來說明好了，看看能不能較容易被你所理解呢！

```
範例一：測試一下是否存在 username 這個變數，若不存在則給予 username 內容為 root
[root@www ~]# echo $username
          <==由於出現空白，所以 username 可能不存在，也可能是空字串
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
root      <==因為 username 沒有設定，所以主動給予名為 root 的內容。
[root@www ~]# username="vbird tsai" <==主動設定 username 的內容
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
vbird tsai <==因為 username 已經設定了，所以使用舊有的設定而不以 root 取代
```

在上面的範例中，重點在於減號『 - 』後面接的關鍵字！基本上你可以這樣理解：

```
new_var=${old_var-content}
   新的變數，主要用來取代舊變數。新舊變數名稱其實常常是一樣的
```

```
new_var=${old_var-content}
   這是本範例中的關鍵字部分！必須要存在的哩！

new_var=${old_var-content}
   舊的變數，被測試的項目！

new_var=${old_var-content}
   變數的『內容』，在本範例中，這個部分是在『給予未設定變數的內容』
```

不過這還是有點問題！因為 username 可能已經被設定為『空字串』了！果真如此的話，那你還可以使用底下的範例來給予 username 的內容成為 root 喔！

```
範例二：若 username 未設定或為空字串，則將 username 內容設定為 root
[root@www ~]# username=""
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
      <==因為 username 被設定為空字串了！所以當然還是保留為空字串！
[root@www ~]# username=${username:-root}
[root@www ~]# echo $username
root  <==加上『 : 』後若變數內容為空或者是未設定，都能夠以後面的內容替換！
```

在大括號內有沒有冒號『 : 』的差別是很大的！加上冒號後，被測試的變數未被設定或者是已被設定為空字串時， 都能夠用後面的內容 (本例中是使用 root 為內容) 來替換與設定！這樣可以瞭解了嗎？除了這樣的測試之外， 還有其他的測試方法喔！鳥哥將他整理如下：

Tips:
底下的例子當中，那個 var 與 str 為變數，我們想要針對 str 是否有設定來決定 var 的值喔！ 一般來說， str: 代表『str 沒設定或為空的字串時』；至於 str 則僅為『沒有該變數』。

變數設定方式	str 沒有設定	str 為空字串	str 已設定非為空字串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=	var=expr	var=expr
var=${str:+expr}	var=	var=	var=expr
var=${str=expr}	str=expr var=expr	str 不變 var=	str 不變 var=$str
var=${str:=expr}	str=expr var=expr	str=expr var=expr	str 不變 var=$str
var=${str?expr}	expr 輸出至 stderr	var=	var=$str
var=${str:?expr}	expr 輸出至 stderr	expr 輸出至 stderr	var=$str

根據上面這張表，我們來進行幾個範例的練習吧！ ^_^！首先讓我們來測試一下，如果舊變數 (str) 不存在時， 我們要給予新變數一個內容，若舊變數存在則新變數內容以舊變數來替換，結果如下：

```
測試：先假設 str 不存在（用 unset），然後測試一下減號（-）的用法：
[root@www ~]# unset str; var=${str-newvar}
[root@www ~]# echo var="$var", str="$str"
```

```
var=newvar, str=          <==因為 str 不存在，所以 var 為 newvar

測試：若 str 已存在，測試一下 var 會變怎樣？：
[root@www ~]# str="oldvar"; var=${str-newvar}
[root@www ~]# echo var="$var", str="$str"
var=oldvar, str=oldvar  <==因為 str 存在，所以 var 等於 str 的內容
```

關於減號 (-) 其實上面我們談過了！這裡的測試只是要讓你更加瞭解，這個減號的測試並不會影響到舊變數的內容。 如果你想要將舊變數內容也一起替換掉的話，那麼就使用等號 (=) 吧！

```
測試：先假設 str 不存在（用 unset），然後測試一下等號（=）的用法：
[root@www ~]# unset str; var=${str=newvar}
[root@www ~]# echo var="$var", str="$str"
var=newvar, str=newvar  <==因為 str 不存在，所以 var/str 均為 newvar

測試：如果 str 已存在了，測試一下 var 會變怎樣？
[root@www ~]# str="oldvar"; var=${str=newvar}
[root@www ~]# echo var="$var", str="$str"
var=oldvar, str=oldvar  <==因為 str 存在，所以 var 等於 str 的內容
```

那如果我只是想知道，如果舊變數不存在時，整個測試就告知我『有錯誤』，此時就能夠使用問號『?』的幫忙啦！ 底下這個測試練習一下先！

```
測試：若 str 不存在時，則 var 的測試結果直接顯示 "無此變數"
[root@www ~]# unset str; var=${str?無此變數}
-bash: str: 無此變數    <==因為 str 不存在，所以輸出錯誤訊息

測試：若 str 存在時，則 var 的內容會與 str 相同！
[root@www ~]# str="oldvar"; var=${str?novar}
[root@www ~]# echo var="$var", str="$str"
var=oldvar, str=oldvar  <==因為 str 存在，所以 var 等於 str 的內容
```

基本上這種變數的測試也能夠透過 shell script 內的 if...then... 來處理， 不過既然 bash 有提供這麼簡單的方法來測試變數，那我們也可以多學一些嘛！ 不過這種變數測試通常是在程式設計當中比較容易出現，如果這裡看不懂就先略過，未來有用到判斷變數值時，再回來看看吧！ ^_^

命令別名與歷史命令：

我們知道在早期的 DOS 年代，清除螢幕上的資訊可以使用 cls 來清除，但是在 Linux 裡面， 我們則是使用 clear 來清除畫面的。那麼可否讓 cls 等於 clear 呢？可以啊！用啥方法？ link file 還是什麼的？別急！底下我們介紹不用 link file 的命令別名來達成。那麼什麼又是歷史命令？ 曾經做過的舉動我們可以將他記錄下來喔！那就是歷史命令囉～底下分別來談一談這兩個玩意兒。

命令別名設定： alias, unalias

命令別名是一個很有趣的東西，特別是你的慣用指令特別長的時候！還有， 增設預設的選項在一些慣用的指令上面，可以預防一些不小心誤殺檔案的情況發生的時候！ 舉個例子來說，如果你要查詢隱藏檔，並且需要長的列出與一頁一頁翻看，那麼需要下達『`ls -al | more`』這個指令，我是覺得很煩啦！ 要輸入好幾個單字！那可不可以使用 lm 來簡化呢？當然可以，你可以在命令列下面下達：

```
[root@www ~]# alias lm='ls -al | more'
```

立刻多出了一個可以執行的指令喔！這個指令名稱為 lm，且其實他是執行 ls -al | more 啊！真是方便。不過， 要注意的是：『alias 的定義規則與變數定義規則幾乎相同』， 所以你只要在 alias 後面加上你的 {『別名』='指令 選項...'}， 以後你只要輸入 lm 就相當於輸入了 ls -al|more 這一串指令！很方便吧！

另外，命令別名的設定還可以取代既有的指令喔！舉例來說，我們知道 root 可以移除 (rm) 任何資料！所以當你以 root 的身份在進行工作時，需要特別小心， 但是總有失手的時候，那麼 rm 提供了一個選項來讓我們確認是否要移除該檔案，那就是 -i 這個選項！所以，你可以這樣做：

```
[root@www ~]# alias rm='rm -i'
```

那麼以後使用 rm 的時候，就不用太擔心會有錯誤刪除的情況了！這也是命令別名的優點囉！ 那麼如何知道目前有哪些的命令別名呢？就使用 alias 呀！

```
[root@www ~]# alias
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias lm='ls -l | more'
alias ls='ls --color=tty'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --show-dot --show-tilde'
```

由上面的資料當中，你也會發現一件事情啊，我們在第十章的 vim 程式編輯器裡面提到 vi 與 vim 是不太一樣的，vim 可以多作一些額外的語法檢驗與顏色顯示，預設的 root 是單純使用 vi 而已。 如果你想要使用 vi 就直接以 vim 來開啟檔案的話，使用『 alias vi='vim' 』這個設定即可。 至於如果要取消命令別名的話，那麼就使用 unalias 吧！例如要將剛剛的 lm 命令別名拿掉，就使用：

```
[root@www ~]# unalias lm
```

那麼命令別名與變數有什麼不同呢？命令別名是『新創一個新的指令， 你可以直接下達該指令』的，至於變數則需要使用類似『 echo 』指令才能夠呼叫出變數的內容！ 這兩者當然不一樣！很多初學者在這裡老是搞不清楚！要注意啊！ ^_^

例題：

DOS 年代，列出目錄與檔案就是 dir，而清除螢幕就是 cls，那麼如果我想要在 linux 裡面也使用相同的指令呢？

答：

很簡單，透過 clear 與 ls 來進行命令別名的建置：

```
alias cls='clear'
alias dir='ls -l'
```

歷史命令：history

前面我們提過 bash 有提供指令歷史的服務！那麼如何查詢我們曾經下達過的指令呢？就使用 history 囉！當然，如果覺得 histsory 要輸入的字元太多太麻煩，可以使用命令別名來設定呢！ 不要跟我說還不會設定呦！ ^_^

```
[root@www ~]# alias h='history'
```

如此則輸入 h 等於輸入 history 囉！好了，我們來談一談 history 的用法吧！

```
[root@www ~]# history [n]
[root@www ~]# history [-c]
[root@www ~]# history [-raw] histfiles
選項與參數：
n    ：數字，意思是『要列出最近的 n 筆命令列表』的意思！
-c   ：將目前的 shell 中的所有 history 內容全部消除
-a   ：將目前新增的 history 指令新增入 histfiles 中，若沒有加 histfiles ，
       則預設寫入 ~/.bash_history
-r   ：將 histfiles 的內容讀到目前這個 shell 的 history 記憶中；
-w   ：將目前的 history 記憶內容寫入 histfiles 中！

範例一：列出目前記憶體內的所有 history 記憶
[root@www ~]# history
# 前面省略
 1017  man bash
 1018  ll
 1019  history
 1020  history
# 列出的資訊當中，共分兩欄，第一欄為該指令在這個 shell 當中的代碼，
# 另一個則是指令本身的內容喔！至於會秀出幾筆指令記錄，則與 HISTSIZE 有關！

範例二：列出目前最近的 3 筆資料
[root@www ~]# history 3
 1019  history
 1020  history
 1021  history 3

範例三：立刻將目前的資料寫入 histfile 當中
[root@www ~]# history -w
# 在預設的情況下，會將歷史紀錄寫入 ~/.bash_history 當中！
[root@www ~]# echo $HISTSIZE
1000
```

在正常的情況下，歷史命令的讀取與記錄是這樣的：

- 當我們以 bash 登入 Linux 主機之後，系統會主動的由家目錄的 ~/.bash_history 讀取以前曾經下過的指令，那麼 ~/.bash_history 會記錄幾筆資料呢？這就與你 bash 的 HISTFILESIZE 這個變數設定值有關了！
- 假設我這次登入主機後，共下達過 100 次指令，『等我登出時， 系統就會將 101~1100 這總共 1000 筆歷史命令**更新**到 ~/.bash_history 當中。』也就是說，歷史命令在我登出時，會將最近的 HISTFILESIZE 筆記錄到我的紀錄檔當中啦！
- 當然，也可以用 history -w 強制立刻寫入的！那為何用『更新』兩個字呢？ 因為 ~/.bash_history 記錄的筆數永遠都是 HISTFILESIZE 那麼多，舊的訊息會被主動的拿掉！ 僅保留最新的！

那麼 history 這個歷史命令只可以讓我查詢命令而已嗎？呵呵！當然不止啊！ 我們可以利用相關的功能來幫我們執行命令呢！舉例來說囉：

```
[root@www ~]# !number
[root@www ~]# !command
[root@www ~]# !!
選項與參數：
number   ：執行第幾筆指令的意思；
command  ：由最近的指令向前搜尋『指令串開頭為 command』的那個指令，並執行；
!!       ：就是執行上一個指令(相當於按↑按鍵後，按 Enter)

[root@www ~]# history
   66  man rm
   67  alias
   68  man history
   69  history
[root@www ~]# !66  <==執行第 66 筆指令
[root@www ~]# !!   <==執行上一個指令，本例中亦即 !66
[root@www ~]# !al  <==執行最近以 al 為開頭的指令(上頭列出的第 67 個)
```

經過上面的介紹，瞭乎？歷史命令用法可多了！如果我想要執行上一個指令， 除了使用上下鍵之外，我可以直接以『 !! 』來下達上個指令的內容，此外， 我也可以直接選擇下達第 n 個指令，『 !n 』來執行，也可以使用指令標頭，例如 『 !vi 』來執行最近指令開頭是 vi 的指令列！相當的方便而好用！

基本上 history 的用途很大的！但是需要小心安全的問題！尤其是 root 的歷史紀錄檔案，這是 Cracker 的最愛！因為不小心的 root 會將很多的重要資料在執行的過程中會被紀錄在 ~/.bash_history 當中，如果這個檔案被解析的話，後果不堪吶！無論如何，使用 history 配合『 ! 』曾經使用過的指令下達是很有效率的一個指令下達方法！

- **同一帳號同時多次登入的 history 寫入問題**

有些朋友在練習 linux 的時候喜歡同時開好幾個 bash 介面，這些 bash 的身份都是 root 。 這樣會有 ~/.bash_history 的寫入問題嗎？想一想，因為這些 bash 在同時以 root 的身份登入， 因此所有的 bash 都有自己的 1000 筆記錄在記憶體中。因為等到登出時才會更新記錄檔，所以囉， 最後登出的那個 bash 才會是最後寫入的資料。唔！如此一來其他 bash 的指令操作就不會被記錄下來了 (其實有被記錄，只是被後來的最後一個 bash 所覆蓋更新了) 。

由於多重登入有這樣的問題，所以很多朋友都習慣單一 bash 登入，再用工作控制 (job control, 第四篇會介紹) 來切換不同工作！ 這樣才能夠將所有曾經下達過的指令記錄下來，也才方便未來系統管理員進行指令的 debug 啊！

- **無法記錄時間**

歷史命令還有一個問題，那就是無法記錄指令下達的時間。由於這 1000 筆歷史命令是依序記錄的，但是並沒有記錄時間，所以在查詢方面會有一些不方便。如果讀者們有興趣，其實可以透過 ~/.bash_logout 來進行 history 的記錄，並加上 date 來增加時間參數，也是一個可以應用的方向喔！ 有興趣的朋友可以先看看情境模擬題一吧！

Bash Shell 的操作環境：

是否記得我們登入主機的時候，螢幕上頭會有一些說明文字，告知我們的 Linux 版本啊什麼的， 還有，

登入的時候我們還可以給予使用者一些訊息或者歡迎文字呢。此外， 我們習慣的環境變數、命令別名等等的，是否可以登入就主動的幫我設定好？ 這些都是需要注意的。另外，這些設定值又可以分為系統整體設定值與各人喜好設定值， 僅是一些檔案放置的地點不同啦！這我們後面也會來談一談的！

路徑與指令搜尋順序

我們在第六章與第七章都曾談過『相對路徑與絕對路徑』的關係， 在本章的前幾小節也談到了 alias 與 bash 的內建命令。現在我們知道系統裡面其實有不少的 ls 指令， 或者是包括內建的 echo 指令，那麼來想一想，如果一個指令 (例如 ls) 被下達時， 到底是哪一個 ls 被拿來運作？很有趣吧！基本上，指令運作的順序可以這樣看：

1. 以相對/絕對路徑執行指令，例如『 /bin/ls 』或『 ./ls 』；
2. 由 alias 找到該指令來執行；
3. 由 bash 內建的 (builtin) 指令來執行；
4. 透過 $PATH 這個變數的順序搜尋到的第一個指令來執行。

舉例來說，你可以下達 /bin/ls 及單純的 ls 看看，會發現使用 ls 有顏色但是 /bin/ls 則沒有顏色。 因為 /bin/ls 是直接取用該指令來下達，而 ls 會因為『 alias ls='ls --color=tty' 』這個命令別名而先使用！ 如果想要瞭解指令搜尋的順序，其實透過 type -a ls 也可以查詢的到啦！上述的順序最好先瞭解喔！

例題：

設定 echo 的命令別名成為 echo -n， 然後再觀察 echo 執行的順序

答：

```
[root@www ~]# alias echo='echo -n'
[root@www ~]# type -a echo
echo is aliased to `echo -n'
echo is a shell builtin
echo is /bin/echo
```

瞧！很清楚吧！先 alias 再 builtin 再由 $PATH 找到 /bin/echo 囉！

---

bash 的進站與歡迎訊息： /etc/issue, /etc/motd

蝦密！ bash 也有進站畫面與歡迎訊息喔？真假？真的啊！ 還記得在終端機介面 (tty1 ~ tty6) 登入的時候，會有幾行提示的字串嗎？那就是進站畫面啊！ 那個字串寫在哪裡啊？呵呵！在 /etc/issue 裡面啊！先來看看：

```
[root@www ~]# cat /etc/issue
CentOS release 5.3 (Final)
Kernel \r on an \m
```

鳥哥是以完全未更新過的 CentOS 5.3 作為範例，裡面預設有三行，較有趣的地方在於 \r 與 \m。 就如同 $PS1 這變數一樣，issue 這個檔案的內容也是可以使用反斜線作為變數取用喔！你可以 man issue 配合 man mingetty 得到底下的結果：

issue 內的各代碼意義

```
\d 本地端時間的日期；
\l 顯示第幾個終端機介面；
\m 顯示硬體的等級（i386/i486/i586/i686...）；
\n 顯示主機的網路名稱；
\o 顯示 domain name；
\r 作業系統的版本（相當於 uname -r）
\t 顯示本地端時間的時間；
\s 作業系統的名稱；
\v 作業系統的版本。
```

做一下底下這個練習，看看能不能取得你要的進站畫面？

例題：

如果你在 tty3 的進站畫面看到如下顯示，該如何設定才能得到如下畫面？

```
CentOS release 5.3 (Final) (terminal: tty3)
Date: 2009-02-05 17:29:19
Kernel 2.6.18-128.el5 on an i686
Welcome!
```

注意，tty3 在不同的 tty 有不同顯示，日期則是再按下 [enter] 後就會所有不同。

答：

很簡單，參考上述的反斜線功能去修改 /etc/issue 成為如下模樣即可(共五行)：

```
CentOS release 5.3 (Final) (terminal: \l)
Date: \d \t
Kernel \r on an \m
Welcome!
```

曾有鳥哥的學生在這個 /etc/issue 內修改資料，光是利用簡單的英文字母作出屬於他自己的進站畫面， 畫面裡面有他的中文名字呢！非常厲害！也有學生做成類似很大一個『囧』在進站畫面，都非常有趣！

你要注意的是，除了 /etc/issue 之外還有個 /etc/issue.net 呢！這是啥？這個是提供給 telnet 這個遠端登入程式用的。 當我們使用 telnet 連接到主機時，主機的登入畫面就會顯示 /etc/issue.net 而不是 /etc/issue 呢！

至於如果您想要讓使用者登入後取得一些訊息，例如您想要讓大家都知道的訊息， 那麼可以將訊息加入 /etc/motd 裡面去！例如：當登入後，告訴登入者， 系統將會在某個固定時間進行維護工作，可以這樣做：

```
[root@www ~]# vi /etc/motd
Hello everyone,
Our server will be maintained at 2009/02/28 0:00 ~ 24:00.
Please don't login server at that time. ^_^
```

那麼當你的使用者(包括所有的一般帳號與 root)登入主機後，就會顯示這樣的訊息出來：

```
Last login: Thu Feb  5 22:35:47 2009 from 127.0.0.1
```

```
Hello everyone,
Our server will be maintained at 2009/02/28 0:00 ~ 24:00.
Please don't login server at that time. ^_^
```

---

bash 的環境設定檔

你是否會覺得奇怪，怎麼我們什麼動作都沒有進行，但是一進入 bash 就取得一堆有用的變數了？ 這是因為系統有一些環境設定檔案的存在，讓 bash 在啟動時直接讀取這些設定檔，以規劃好 bash 的操作環境啦！ 而這些設定檔又可以分為全體系統的設定檔以及使用者個人偏好設定檔。要注意的是，我們前幾個小節談到的命令別名啦、自訂的變數啦，在你登出 bash 後就會失效，所以你想要保留你的設定， 就得要將這些設定寫入設定檔才行。底下就讓我們來聊聊吧！

---

login 與 non-login shell

在開始介紹 bash 的設定檔前，我們一定要先知道的就是 login shell 與 non-login shell！ 重點在於有沒有登入 (login) 啦！

- login shell：取得 bash 時需要完整的登入流程的，就稱為 login shell。舉例來說，你要由 tty1 ~ tty6 登入，需要輸入使用者的帳號與密碼，此時取得的 bash 就稱為『 login shell 』囉；
- non-login shell：取得 bash 介面的方法不需要重複登入的舉動，舉例來說，(1)你以 X window 登入 Linux 後， 再以 X 的圖形化介面啟動終端機，此時那個終端介面並沒有需要再次的輸入帳號與密碼，那個 bash 的環境就稱為 non-login shell了。(2)你在原本的 bash 環境下再次下達 bash 這個指令，同樣的也沒有輸入帳號密碼， 那第二個 bash（子程序）也是 non-login shell 。

為什麼要介紹 login, non-login shell 呢？這是因為這兩個取得 bash 的情況中，讀取的設定檔資料並不一樣所致。 由於我們需要登入系統，所以先談談 login shell 會讀取哪些設定檔？一般來說，login shell 其實只會讀取這兩個設定檔：

1. /etc/profile：這是系統整體的設定，你最好不要修改這個檔案；
2. ~/.bash_profile 或 ~/.bash_login 或 ~/.profile：屬於使用者個人設定，你要改自己的資料，就寫入這裡！

那麼，就讓我們來聊一聊這兩個檔案吧！這兩個檔案的內容可是非常繁複的喔！

/etc/profile (login shell 才會讀)

你可以使用 vim 去閱讀一下這個檔案的內容。這個設定檔可以利用使用者的識別碼 (UID) 來決定很多重要的變數資料， 這也是每個使用者登入取得 bash 時一定會讀取的設定檔！ 所以如果你想要幫所有使用者設定整體環境，那就是改這裡囉！不過，沒事還是不要隨便改這個檔案喔 這個檔案設定的變數主要有：

- PATH：會依據 UID 決定 PATH 變數要不要含有 sbin 的系統指令目錄；
- MAIL：依據帳號設定好使用者的 mailbox 到 /var/spool/mail/帳號名；
- USER：根據使用者的帳號設定此一變數內容；
- HOSTNAME：依據主機的 hostname 指令決定此一變數內容；
- HISTSIZE：歷史命令記錄筆數。CentOS 5.x 設定為 1000 ；

/etc/profile 可不止會做這些事而已，他還會去呼叫外部的設定資料喔！在 CentOS 5.x 預設的情況下，底下這些資料會依序的被呼叫進來：

- /etc/inputrc

  其實這個檔案並沒有被執行啦！/etc/profile 會主動的判斷使用者有沒有自訂輸入的按鍵功能，如果沒有的話， /etc/profile 就會決定設定『INPUTRC=/etc/inputrc』這個變數！此一檔案內容為 bash 的熱鍵啦、[tab]要不要有聲音啦等等的資料！ 因為鳥哥覺得 bash 預設的環境已經很棒了，所以不建議修改這個檔案！

- /etc/profile.d/*.sh

  其實這是個目錄內的眾多檔案！只要在 /etc/profile.d/ 這個目錄內且副檔名為 .sh ， 另外，使用者能夠具有 r 的權限， 那麼該檔案就會被 /etc/profile 呼叫進來。在 CentOS 5.x 中，這個目錄底下的檔案規範了 bash 操作介面的顏色、 語系、ll 與 ls 指令的命令別名、vi 的命令別名、which 的命令別名等等。如果你需要幫所有使用者設定一些共用的命令別名時， 可以在這個目錄底下自行建立副檔名為 .sh 的檔案，並將所需要的資料寫入即可喔！

- /etc/sysconfig/i18n

  這個檔案是由 /etc/profile.d/lang.sh 呼叫進來的！這也是我們決定 bash 預設使用何種語系的重要設定檔！ 檔案裡最重要的就是 LANG 這個變數的設定啦！我們在前面的 locale 討論過這個檔案囉！ 自行回去瞧瞧先！

反正你只要記得，bash 的 login shell 情況下所讀取的整體環境設定檔其實只有 /etc/profile，但是 /etc/profile 還會呼叫出其他的設定檔，所以讓我們的 bash 操作介面變的非常的友善啦！ 接下來，讓我們來瞧瞧，那麼個人偏好的設定檔又是怎麼回事？

~/.bash_profile (login shell 才會讀)

bash 在讀完了整體環境設定的 /etc/profile 並藉此呼叫其他設定檔後，接下來則是會讀取使用者的個人設定檔。 在 login shell 的 bash 環境中，所讀取的個人偏好設定檔其實主要有三個，依序分別是：

1. ~/.bash_profile
2. ~/.bash_login
3. ~/.profile

其實 bash 的 login shell 設定只會讀取上面三個檔案的其中一個， 而讀取的順序則是依照上面的順序。也就是說，如果 ~/.bash_profile 存在，那麼其他兩個檔案不論有無存在，都不會被讀取。 如果 ~/.bash_profile 不存在才會去讀取 ~/.bash_login，而前兩者都不存在才會讀取 ~/.profile 的意思。 會有這麼多的檔案，其實是因應其他 shell 轉換過來的使用者的習慣而已。 先讓我們來看一下 root 的 /root/.bash_profile 的內容是怎樣呢？

```
[root@www ~]# cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then    <==底下這三行在判斷並讀取 ~/.bashrc
        . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin         <==底下這幾行在處理個人化設定
export PATH
unset USERNAME
```

這個檔案內有設定 PATH 這個變數喔！而且還使用了 export 將 PATH 變成環境變數呢！ 由於 PATH 在 /etc/profile 當中已經設定過，所以在這裡就以累加的方式增加使用者家目錄下的 ~/bin/ 為額外的執行檔放置目錄。這也就是說，你可以將自己建立的執行檔放置到你自己家目錄下的 ~/bin/ 目錄啦！ 那

就可以直接執行該執行檔而不需要使用絕對/相對路徑來執行該檔案。

這個檔案的內容比較有趣的地方在於 if ... then ... 那一段！那一段程式碼我們會在第十三章 shell script 談到，假設你現在是看不懂的。該段的內容指的是『判斷家目錄下的 ~/.bashrc 存在否，若存在則讀入 ~/.bashrc 的設定』。bash 設定檔的讀入方式比較有趣，主要是透過一個指令『 source 』來讀取的！也就是說 ~/.bash_profile 其實會再呼叫 ~/.bashrc 的設定內容喔！最後，我們來看看整個 login shell 的讀取流程：

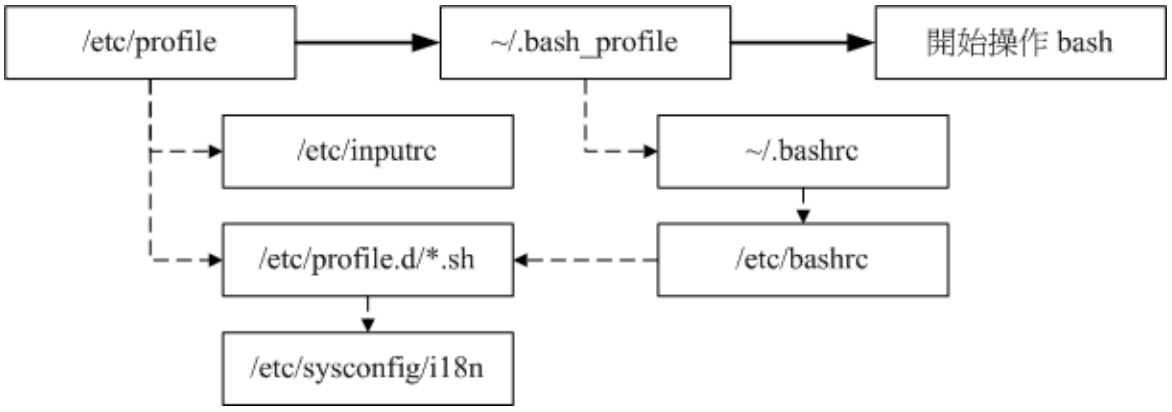

圖 4.3.1、login shell 的設定檔讀取流程

實線的的方向是主線流程，虛線的方向則是被呼叫的設定檔！從上面我們也可以清楚的知道，在 CentOS 的 login shell 環境下，最終被讀取的設定檔是『 ~/.bashrc 』這個檔案喔！所以，你當然可以將自己的偏好設定寫入該檔案即可。底下我們還要討論一下 source 與 ~/.bashrc 喔！

- **source ：讀入環境設定檔的指令**

由於 /etc/profile 與 ~/.bash_profile 都是在取得 login shell 的時候才會讀取的設定檔，所以，如果你將自己的偏好設定寫入上述的檔案後，通常都是得登出再登入後，該設定才會生效。那麼，能不能直接讀取設定檔而不登出登入呢？可以的！那就得要利用 source 這個指令了！

```
[root@www ~]# source 設定檔檔名

範例：將家目錄的 ~/.bashrc 的設定讀入目前的 bash 環境中
[root@www ~]# source ~/.bashrc  <==底下這兩個指令是一樣的！
[root@www ~]#  .  ~/.bashrc
```

利用 source 或小數點 (.) 都可以將設定檔的內容讀進來目前的 shell 環境中！舉例來說，我修改了 ~/.bashrc ，那麼不需要登出，立即以 source ~/.bashrc 就可以將剛剛最新設定的內容讀進來目前的環境中！很不錯吧！還有，包括 ~/bash_profile 以及 /etc/profile 的設定中，很多時候也都是利用到這個 source (或小數點) 的功能喔！

有沒有可能會使用到不同環境設定檔的時候？有啊！最常發生在一個人的工作環境分為多種情況的時候了！舉個例子來說，在鳥哥的大型主機中，常常需要負責兩到三個不同的案子，每個案子所需要處理的環境變數訂定並不相同，那麼鳥哥就將這兩三個案子分別編寫屬於該案子的環境變數設定檔案，當需要該環境時，就直接『 source 變數檔 』，如此一來，環境變數的設定就變的更簡便而靈活了！

- **~/.bashrc (non-login shell 會讀)**

談完了 login shell 後，那麼 non-login shell 這種非登入情況取得 bash 操作介面的環境設定檔又是什麼？當你取得 non-login shell 時，該 bash 設定檔僅會讀取 ~/.bashrc 而已啦！那麼預設的 ~/.bashrc 內容是如何？

```
[root@www ~]# cat ~/.bashrc
# .bashrc

# User specific aliases and functions
alias rm='rm -i'              <==使用者的個人設定
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then   <==整體的環境設定
        . /etc/bashrc
fi
```

特別注意一下，由於 root 的身份與一般使用者不同，鳥哥是以 root 的身份取得上述的資料， 如果是一般使用者的 ~/.bashrc 會有些許不同。看一下，你會發現在 root 的 ~/.bashrc 中其實已經規範了較為保險的命令別名了。 此外，咱們的 CentOS 5.x 還會主動的呼叫 /etc/bashrc 這個檔案喔！為什麼需要呼叫 /etc/bashrc 呢？ 因為 /etc/bashrc 幫我們的 bash 定義出底下的資料：

- 依據不同的 UID 規範出 umask 的值；
- 依據不同的 UID 規範出提示字元（就是 PS1 變數）；
- 呼叫 /etc/profile.d/*.sh 的設定

你要注意的是，這個 /etc/bashrc 是 CentOS 特有的 (其實是 Red Hat 系統特有的)，其他不同的 distributions 可能會放置在不同的檔名就是了。由於這個 ~/.bashrc 會呼叫 /etc/bashrc 及 /etc/profile.d/*.sh， 所以，萬一你沒有 ~/.bashrc (可能自己不小心將他刪除了)，那麼你會發現你的 bash 提示字元可能會變成這個樣子：

```
-bash-3.2$
```

不要太擔心啦！這是正常的，因為你並沒有呼叫 /etc/bashrc 來規範 PS1 變數啦！而且這樣的情況也不會影響你的 bash 使用。 如果你想要將命令提示字元捉回來，那麼可以複製 /etc/skel/.bashrc 到你的家目錄，再修訂一下你所想要的內容， 並使用 source 去呼叫 ~/.bashrc ， 那你的命令提示字元就會回來啦！

- **其他相關設定檔**

事實上還有一些設定檔可能會影響到你的 bash 操作的，底下就來談一談：

- **/etc/man.config**

  這個檔案乍看之下好像跟 bash 沒相關性，但是對於系統管理員來說， 卻也是很重要的一個檔案！這的檔案的內容『規範了使用 man 的時候， man page 的路徑到哪裡去尋找！』所以說的簡單一點，這個檔案規定了下達 man 的時候，該去哪裡查看資料的路徑設定！

  那麼什麼時候要來修改這個檔案呢？如果你是以 tarball 的方式來安裝你的資料，那麼你的 man page 可能會放置在 /usr/local/softpackage/man 裡頭，那個 softpackage 是你的套件名稱， 這個時候你就得以手動的方式將該路徑加到 /etc/man.config 裡頭，否則使用 man 的時候就會找不到相關的說明檔囉。

  事實上，這個檔案內最重要的其實是 MANPATH 這個變數設定啦！ 我們搜尋 man page 時，會依據 MANPATH 的路徑去分別搜尋啊！另外，要注意的是， 這個檔案在各大不同版本 Linux distributions 中，檔名都不太相同，例如 CentOS 用的是 /etc/man.config ， 而 SuSE 用的則是 /etc/manpath.config ， 可以利用 [tab] 按鍵來進行檔名的補齊啦！

- **~/.bash_history**

還記得我們在歷史命令提到過這個檔案吧？預設的情況下， 我們的歷史命令就記錄在這裡啊！而這個檔案能夠記錄幾筆資料，則與 HISTFILESIZE 這個變數有關啊。每次登入 bash 後，bash 會先讀取這個檔案，將所有的歷史指令讀入記憶體， 因此，當我們登入 bash 後就可以查知上次使用過哪些指令囉。至於更多的歷史指令， 請自行回去參考喔！

- **~/.bash_logout**

  這個檔案則記錄了『當我登出 bash 後，系統再幫我做完什麼動作後才離開』的意思。 你可以去讀取一下這個檔案的內容，預設的情況下，登出時， bash 只是幫我們清掉螢幕的訊息而已。 不過，你也可以將一些備份或者是其他你認為重要的工作寫在這個檔案中 (例如清空暫存檔)， 那麼當你離開 Linux 的時候，就可以解決一些煩人的事情囉！

終端機的環境設定： stty, set

我們在第五章首次登入 Linux 時就提過，可以在 tty1 ~ tty6 這六個文字介面的終端機 (terminal) 環境中登入，登入的時候我們可以取得一些字元設定的功能喔！ 舉例來說，我們可以利用倒退鍵 (backspace，就是那個←符號的按鍵) 來刪除命令列上的字元， 也可以使用 [ctrl]+c 來強制終止一個指令的運行，當輸入錯誤時，就會有聲音跑出來警告。這是怎麼辦到的呢？ 很簡單啊！因為登入終端機的時候，會自動的取得一些終端機的輸入環境的設定啊！

事實上，目前我們使用的 Linux distributions 都幫我們作了最棒的使用者環境了， 所以大家可以不用擔心操作環境的問題。不過，在某些 Unix like 的機器中，還是可能需要動用一些手腳， 才能夠讓我們的輸入比較快樂～舉例來說，利用 [backspace] 刪除，要比利用 [Del] 按鍵來的順手吧！ 但是某些 Unix 偏偏是以 [del] 來進行字元的刪除啊！所以，這個時候就可以動動手腳囉～

那麼如何查閱目前的一些按鍵內容呢？可以利用 stty (setting tty 終端機的意思) 呢！ stty 也可以幫助設定終端機的輸入按鍵代表意義喔！

```
[root@www ~]# stty [-a]
選項與參數：
-a   ：將目前所有的 stty 參數列出來；

範例一：列出所有的按鍵與按鍵內容
[root@www ~]# stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
....(以下省略)....
```

我們可以利用 stty -a 來列出目前環境中所有的按鍵列表，在上頭的列表當中，需要注意的是特殊字體那幾個， 此外，如果出現 ^ 表示 [Ctrl] 那個按鍵的意思。舉例來說， intr = ^C 表示利用 [ctrl] + c 來達成的。幾個重要的代表意義是：

- eof   ：End of file 的意思，代表『結束輸入』。
- erase ：向後刪除字元，
- intr ：送出一個 interrupt（中斷）的訊號給目前正在 run 的程序；
- kill ：刪除在目前指令列上的所有文字；
- quit ：送出一個 quit 的訊號給目前正在 run 的程序；
- start ：在某個程序停止後，重新啟動他的 output
- stop ：停止目前螢幕的輸出；
- susp ：送出一個 terminal stop 的訊號給正在 run 的程序。

記不記得我們在第五章講過幾個 Linux 熱鍵啊？沒錯！ 就是這個 stty 設定值內的 intr / eof 囉～至於刪

除字元，就是 erase 那個設定值啦！ 如果你想要用 [ctrl]+h 來進行字元的刪除，那麼可以下達：

```
[root@www ~]# stty erase ^h
```

那麼從此之後，你的刪除字元就得要使用 [ctrl]+h 囉，按下 [backspace] 則會出現 ^? 字樣呢！ 如果想要回復利用 [backspace]，就下達 stty erase ^? 即可啊！ 至於更多的 stty 說明，記得參考一下 man stty 的內容喔！

除了 stty 之外，其實我們的 bash 還有自己的一些終端機設定值呢！那就是利用 set 來設定的！ 我們之前提到一些變數時，可以利用 set 來顯示，除此之外，其實 set 還可以幫我們設定整個指令輸出/輸入的環境。 例如記錄歷史命令、顯示錯誤內容等等。

```
[root@www ~]# set [-uvCHhmBx]
選項與參數：
-u   ：預設不啟用。若啟用後，當使用未設定變數時，會顯示錯誤訊息；
-v   ：預設不啟用。若啟用後，在訊息被輸出前，會先顯示訊息的原始內容；
-x   ：預設不啟用。若啟用後，在指令被執行前，會顯示指令內容(前面有 ++ 符號)
-h   ：預設啟用。與歷史命令有關；
-H   ：預設啟用。與歷史命令有關；
-m   ：預設啟用。與工作管理有關；
-B   ：預設啟用。與刮號 [] 的作用有關；
-C   ：預設不啟用。若使用 > 等，則若檔案存在時，該檔案不會被覆蓋。

範例一：顯示目前所有的 set 設定值
[root@www ~]# echo $-
himBH
# 那個 $- 變數內容就是 set 的所有設定啦！ bash 預設是 himBH 喔！

範例二：設定 "若使用未定義變數時，則顯示錯誤訊息"
[root@www ~]# set -u
[root@www ~]# echo $vbirding
-bash: vbirding: unbound variable
# 預設情況下，未設定/未宣告 的變數都會是『空的』，不過，若設定 -u 參數，
# 那麼當使用未設定的變數時，就會有問題啦！很多的 shell 都預設啟用 -u 參數。
# 若要取消這個參數，輸入 set +u 即可！

範例三：執行前，顯示該指令內容。
[root@www ~]# set -x
[root@www ~]# echo $HOME
+ echo /root
/root
++ echo -ne '\033]0;root@www:~'
# 看見否？要輸出的指令都會先被列印到螢幕上喔！前面會多出 + 的符號！
```

另外，其實我們還有其他的按鍵設定功能呢！就是在前一小節提到的 /etc/inputrc 這個檔案裡面設定。

```
[root@www ~]# cat /etc/inputrc
# do not bell on tab-completion
#set bell-style none

set meta-flag on
set input-meta on
set convert-meta off
set output-meta on
.....以下省略.....
```

還有例如 /etc/DIR_COLORS* 與 /etc/termcap 等，也都是與終端機有關的環境設定檔案呢！ 不過，事實上，鳥哥並不建議您修改 tty 的環境呢，這是因為 bash 的環境已經設定的很親和了， 我們不需要額外的設定或者修改，否則反而會產生一些困擾。不過，寫在這裡的資料， 只是希望大家能夠清楚的知道我們的終端機是如何進行設定的喔！ ^_^！ 最後，我們將 bash 預設的組合鍵給他彙整如下：

組合按鍵	執行結果
Ctrl + C	終止目前的命令
Ctrl + D	輸入結束 (EOF)，例如郵件結束的時候；
Ctrl + M	就是 Enter 啦！
Ctrl + S	暫停螢幕的輸出
Ctrl + Q	恢復螢幕的輸出
Ctrl + U	在提示字元下，將整列命令刪除
Ctrl + Z	『暫停』目前的命令

萬用字元與特殊符號

在 bash 的操作環境中還有一個非常有用的功能，那就是萬用字元 (wildcard) ！ 我們利用 bash 處理資料就更方便了！底下我們列出一些常用的萬用字元喔：

符號	意義
*	代表『 0 個到無窮多個』任意字元
?	代表『一定有一個』任意字元
[ ]	同樣代表『一定有一個在括號內』的字元(非任意字元)。例如 [abcd] 代表『一定有一個字元， 可能是 a, b, c, d 這四個任何一個』
[-]	若有減號在中括號內時，代表『在編碼順序內的所有字元』。例如 [0-9] 代表 0 到 9 之間的所有數字，因為數字的語系編碼是連續的！
[^]	若中括號內的第一個字元為指數符號 (^)，那表示『反向選擇』，例如 [^abc] 代表 一定有一個字元，只要是非 a, b, c 的其他字元就接受的意思。

接下來讓我們利用萬用字元來玩些東西吧！首先，利用萬用字元配合 ls 找檔名看看：

```
[root@www ~]# LANG=C                <==由於與編碼有關，先設定語系一下

範例一：找出 /etc/ 底下以 cron 為開頭的檔名
[root@www ~]# ll -d /etc/cron*      <==加上 -d 是為了僅顯示目錄而已

範例二：找出 /etc/ 底下檔名『剛好是五個字母』的檔名
[root@www ~]# ll -d /etc/?????      <==由於 ? 一定有一個，所以五個 ? 就對了

範例三：找出 /etc/ 底下檔名含有數字的檔名
[root@www ~]# ll -d /etc/*[0-9]*    <==記得中括號左右兩邊均需 *

範例四：找出 /etc/ 底下，檔名開頭非為小寫字母的檔名：
[root@www ~]# ll -d /etc/[^a-z]*    <==注意中括號左邊沒有 *

範例五：將範例四找到的檔案複製到 /tmp 中
[root@www ~]# cp -a /etc/[^a-z]* /tmp
```

除了萬用字元之外，bash 環境中的特殊符號有哪些呢？底下我們先彙整一下：

符號	內容
#	註解符號：這個最常被使用在 script 當中，視為說明！在後的資料均不執行
\	跳脫符號：將『特殊字元或萬用字元』還原成一般字元
|	管線 (pipe)：分隔兩個管線命令的界定(後兩節介紹)；
;	連續指令下達分隔符號：連續性命令的界定 (注意！與管線命令並不相同)
~	使用者的家目錄
$	取用變數前置字元：亦即是變數之前需要加的變數取代值
&	工作控制 (job control)：將指令變成背景下工作
!	邏輯運算意義上的『非』 not 的意思！
/	目錄符號：路徑分隔的符號
>, >>	資料流重導向：輸出導向，分別是『取代』與『累加』
<, <<	資料流重導向：輸入導向 (這兩個留待下節介紹)
' '	單引號，不具有變數置換的功能
" "	具有變數置換的功能！
` `	兩個『 ` 』中間為可以先執行的指令，亦可使用 $( )
( )	在中間為子 shell 的起始與結束
{ }	在中間為命令區塊的組合！

以上為 bash 環境中常見的特殊符號彙整！理論上，你的『檔名』盡量不要使用到上述的字元啦！

資料流重導向

資料流重導向 (redirect) 由字面上的意思來看，好像就是將『資料給他傳導到其他地方去』的樣子？ 沒錯～資料流重導向就是將某個指令執行後應該要出現在螢幕上的資料， 給他傳輸到其他的地方，例如檔案或者是裝置 (例如印表機之類的)！這玩意兒在 Linux 的文字模式底下可重要的！ 尤其是如果我們想要將某些資料儲存下來時，就更有用了！

什麼是資料流重導向

什麼是資料流重導向啊？這得要由指令的執行結果談起！一般來說，如果你要執行一個指令，通常他會是這樣的：

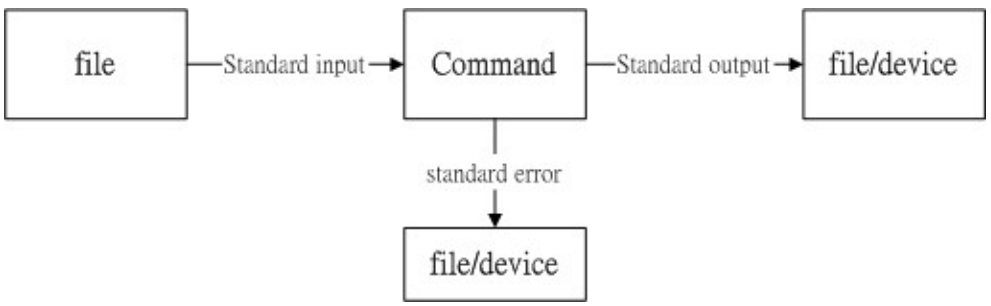

圖 5.1.1、指令執行過程的資料傳輸情況

我們執行一個指令的時候，這個指令可能會由檔案讀入資料，經過處理之後，再將資料輸出到螢幕

上。 在上圖當中， standard output 與 standard error output 分別代表『標準輸出』與『標準錯誤輸出』， 這兩個玩意兒預設都是輸出到螢幕上面來的啊！那麼什麼是標準輸出與標準錯誤輸出呢？

- **standard output 與 standard error output**

簡單的說，標準輸出指的是『指令執行所回傳的正確的訊息』，而標準錯誤輸出可理解為『 指令執行失敗後，所回傳的錯誤訊息』。舉個簡單例子來說，我們的系統預設有 /etc/crontab 但卻無 /etc/vbirdsay， 此時若下達『 cat /etc/crontab /etc/vbirdsay 』這個指令時，cat 會進行：

- 標準輸出：讀取 /etc/crontab 後，將該檔案內容顯示到螢幕上；
- 標準錯誤輸出：因為無法找到 /etc/vbirdsay，因此在螢幕上顯示錯誤訊息

不管正確或錯誤的資料都是預設輸出到螢幕上，所以螢幕當然是亂亂的！那能不能透過某些機制將這兩股資料分開呢？ 當然可以啊！那就是資料流重導向的功能啊！資料流重導向可以將 standard output (簡稱 stdout) 與 standard error output (簡稱 stderr) 分別傳送到其他的檔案或裝置去，而分別傳送所用的特殊字元則如下所示：

1. 標準輸入　(stdin)：代碼為 0 ，使用 < 或 << ；
2. 標準輸出　(stdout)：代碼為 1 ，使用 > 或 >> ；
3. 標準錯誤輸出(stderr)：代碼為 2 ，使用 2> 或 2>> ；

為了理解 stdout 與 stderr， 我們先來進行一個範例的練習：

```
範例一：觀察你的系統根目錄（/）下各目錄的檔名、權限與屬性，並記錄下來
[root@www ~]# ll /  <==此時螢幕會顯示出檔名資訊

[root@www ~]# ll / > ~/rootfile <==螢幕並無任何資訊
[root@www ~]# ll  ~/rootfile <==有個新檔被建立了！
-rw-r--r-- 1 root root 1089 Feb  6 17:00 /root/rootfile
```

怪了！螢幕怎麼會完全沒有資料呢？這是因為原本『 ll / 』所顯示的資料已經被重新導向到 ~/rootfile 檔案中了！ 那個 ~/rootfile 的檔名可以隨便你取。如果你下達『 cat ~/rootfile 』那就可以看到原本應該在螢幕上面的資料囉。 如果我再次下達：『 ll /home > ~/rootfile 』後，那個 ~/rootfile 檔案的內容變成什麼？ 他將變成『僅有 ll /home 的資料』而已！咦！原本的『 ll / 』資料就不見了嗎？是的！因為該檔案的建立方式是：

1. 該檔案（本例中是 ~/rootfile）若不存在，系統會自動的將他建立起來，但是
2. 當這個檔案存在的時候，那麼系統就會先將這個檔案內容清空，然後再將資料寫入！
3. 也就是若以 > 輸出到一個已存在的檔案中，那個檔案就會被覆蓋掉囉！

那如果我想要將資料累加而不想要將舊的資料刪除，那該如何是好？利用兩個大於的符號 (>>) 就好啦！以上面的範例來說，你應該要改成『 ll / >> ~/rootfile 』即可。 如此一來，當 (1) ~/rootfile 不存在時系統會主動建立這個檔案；(2)若該檔案已存在， 則資料會在該檔案的最下方累加進去！

上面談到的是 standard output 的正確資料，那如果是 standard error output 的錯誤資料呢？那就透過 2> 及 2>> 囉！同樣是覆蓋 (2>) 與累加 (2>>) 的特性！我們在剛剛才談到 stdout 代碼是 1 而 stderr 代碼是 2， 所以這個 2> 是很容易理解的，而如果僅存在 > 時，則代表預設的代碼 1 囉！也就是說：

- 1> ：以覆蓋的方法將『正確的資料』輸出到指定的檔案或裝置上；
- 1>>：以累加的方法將『正確的資料』輸出到指定的檔案或裝置上；
- 2> ：以覆蓋的方法將『錯誤的資料』輸出到指定的檔案或裝置上；
- 2>>：以累加的方法將『錯誤的資料』輸出到指定的檔案或裝置上；

要注意喔，『 1>> 』以及『 2>> 』中間是沒有空格的！OK！有些概念之後讓我們繼續聊一聊這傢伙怎麼應用吧！ 當你以一般身份執行 find 這個指令的時候，由於權限的問題可能會產生一些錯誤資訊。例如執行『 find / -name testing 』時，可能會產生類似『 find: /root: Permission denied 』之類的訊

息。 例如底下這個範例：

```
範例二：利用一般身份帳號搜尋 /home 底下是否有名為 .bashrc 的檔案存在
[root@www ~]# su - dmtsai  <==假設我的系統有名為 dmtsai 的帳號
[dmtsai@www ~]$ find /home -name .bashrc <==身份是 dmtsai 喔！
find: /home/lost+found: Permission denied  <== Standard error
find: /home/alex: Permission denied        <== Standard error
find: /home/arod: Permission denied        <== Standard error
/home/dmtsai/.bashrc                       <== Standard output
```

由於 /home 底下還有我們之前建立的帳號存在，那些帳號的家目錄你當然不能進入啊！所以就會有錯誤及正確資料了。 好了，那麼假如我想要將資料輸出到 list 這個檔案中呢？執行『 find /home -name .bashrc > list 』 會有什麼結果？呵呵，你會發現 list 裡面存了剛剛那個『正確』的輸出資料， 至於螢幕上還是會有錯誤的訊息出現呢！傷腦筋！如果想要將正確的與錯誤的資料分別存入不同的檔案中需要怎麼做？

```
範例三：承範例二，將 stdout 與 stderr 分存到不同的檔案去
[dmtsai@www ~]$ find /home -name .bashrc > list_right 2> list_error
```

注意喔，此時『螢幕上不會出現任何訊息』！因為剛剛執行的結果中，有 Permission 的那幾行錯誤資訊都會跑到 list_error 這個檔案中，至於正確的輸出資料則會存到 list_right 這個檔案中囉！這樣可以瞭解了嗎？ 如果有點混亂的話，去休息一下再來看看吧！

- **/dev/null 垃圾桶黑洞裝置與特殊寫法**

想像一下，如果我知道錯誤訊息會發生，所以要將錯誤訊息忽略掉而不顯示或儲存呢？ 這個時候黑洞裝置 /dev/null 就很重要了！這個 /dev/null 可以吃掉任何導向這個裝置的資訊喔！將上述的範例修訂一下：

```
範例四：承範例三，將錯誤的資料丟棄，螢幕上顯示正確的資料
[dmtsai@www ~]$ find /home -name .bashrc 2> /dev/null
/home/dmtsai/.bashrc  <==只有 stdout 會顯示到螢幕上， stderr 被丟棄了
```

再想像一下，如果我要將正確與錯誤資料通通寫入同一個檔案去呢？這個時候就得要使用特殊的寫法了！ 我們同樣用底下的案例來說明：

```
範例五：將指令的資料全部寫入名為 list 的檔案中
[dmtsai@www ~]$ find /home -name .bashrc > list 2> list  <==錯誤
[dmtsai@www ~]$ find /home -name .bashrc > list 2>&1     <==正確
[dmtsai@www ~]$ find /home -name .bashrc &> list         <==正確
```

上述表格第一行錯誤的原因是，由於兩股資料同時寫入一個檔案，又沒有使用特殊的語法， 此時兩股資料可能會交叉寫入該檔案內，造成次序的錯亂。所以雖然最終 list 檔案還是會產生，但是裡面的資料排列就會怪怪的，而不是原本螢幕上的輸出排序。 至於寫入同一個檔案的特殊語法如上表所示，你可以使用 2>&1 也可以使用 &> ！ 一般來說，鳥哥比較習慣使用 2>&1 的語法啦！

- **standard input ： < 與 <<**

瞭解了 stderr 與 stdout 後，那麼那個 < 又是什麼呀？呵呵！以最簡單的說法來說， 那就是『將原本需要由鍵盤輸入的資料，改由檔案內容來取代』的意思。 我們先由底下的 cat 指令操作來瞭解一下什麼叫做『鍵盤輸入』吧！

```
範例六：利用 cat 指令來建立一個檔案的簡單流程
[root@www ~]# cat > catfile
testing
cat file test
<==這裡按下 [ctrl]+d 來離開

[root@www ~]# cat catfile
testing
cat file test
```

由於加入 > 在 cat 後，所以那個 catfile 會被主動的建立，而內容就是剛剛鍵盤上面輸入的那兩行資料了。 唔！那我能不能用純文字檔取代鍵盤的輸入，也就是說，用某個檔案的內容來取代鍵盤的敲擊呢？ 可以的！如下所示：

```
範例七：用 stdin 取代鍵盤的輸入以建立新檔案的簡單流程
[root@www ~]# cat > catfile < ~/.bashrc
[root@www ~]# ll catfile ~/.bashrc
-rw-r--r-- 1 root root 194 Sep 26 13:36 /root/.bashrc
-rw-r--r-- 1 root root 194 Feb  6 18:29 catfile
# 注意看，這兩個檔案的大小會一模一樣！幾乎像是使用 cp 來複製一般！
```

這東西非常的有幫助！尤其是用在類似 mail 這種指令的使用上。 理解 < 之後，再來則是怪可怕一把的 << 這個連續兩個小於的符號了。 他代表的是『結束的輸入字元』的意思！舉例來講：『我要用 cat 直接將輸入的訊息輸出到 catfile 中， 且當由鍵盤輸入 eof 時，該次輸入就結束』，那我可以這樣做：

```
[root@www ~]# cat > catfile << "eof"
> This is a test.
> OK now stop
> eof  <==輸入這關鍵字，立刻就結束而不需要輸入 [ctrl]+d

[root@www ~]# cat catfile
This is a test.
OK now stop     <==只有這兩行，不會存在關鍵字那一行！
```

看到了嗎？利用 << 右側的控制字元，我們可以終止一次輸入， 而不必輸入 [crtl]+d 來結束哩！這對程式寫作很有幫助喔！好了，那麼為何要使用命令輸出重導向呢？我們來說一說吧！

- 螢幕輸出的資訊很重要，而且我們需要將他存下來的時候；
- 背景執行中的程式，不希望他干擾螢幕正常的輸出結果時；
- 一些系統的例行命令（例如寫在 /etc/crontab 中的檔案）的執行結果，希望他可以存下來時；
- 一些執行命令的可能已知錯誤訊息時，想以『 2> /dev/null 』將他丟掉時；
- 錯誤訊息與正確訊息需要分別輸出時。

當然還有很多的功能的，最簡單的就是網友們常常問到的：『為何我的 root 都會收到系統 crontab 寄來的錯誤訊息呢』這個咚咚是常見的錯誤， 而如果我們已經知道這個錯誤訊息是可以忽略的時候，嗯！『 2> errorfile 』這個功能就很重要了吧！ 瞭解了嗎？

命令執行的判斷依據： ; , &&, ||

在某些情況下，很多指令我想要一次輸入去執行，而不想要分次執行時，該如何是好？基本上你有兩個選擇， 一個是透過第十三章要介紹的 shell script 撰寫腳本去執行，一種則是透過底下的介紹來一次輸入多重指令喔！

---

cmd ; cmd（不考慮指令相關性的連續指令下達）

在某些時候，我們希望可以一次執行多個指令，例如在關機的時候我希望可以先執行兩次 sync 同步化寫入磁碟後才 shutdown 電腦，那麼可以怎麼作呢？這樣做呀：

```
[root@www ~]# sync; sync; shutdown -h now
```

在指令與指令中間利用分號 (;) 來隔開，這樣一來，分號前的指令執行完後就會立刻接著執行後面的指令了。 這真是方便啊～再來，換個角度來想，萬一我想要在某個目錄底下建立一個檔案，也就是說，如果該目錄存在的話， 那我才建立這個檔案，如果不存在，那就算了。也就是說這兩個指令彼此之間是有相關性的， 前一個指令是否成功的執行與後一個指令是否要執行有關！那就得動用到 && 或 || 囉！

$?（指令回傳值）與 && 或 ||

如同上面談到的，兩個指令之間有相依性，而這個相依性主要判斷的地方就在於前一個指令執行的結果是否正確。 還記得本章之前我們曾介紹過指令回傳值吧！嘿嘿！沒錯，您真聰明！就是透過這個回傳值啦！ 再複習一次『若前一個指令執行的結果為正確，在 Linux 底下會回傳一個 $? = 0 的值』。 那麼我們怎麼透過這個回傳值來判斷後續的指令是否要執行呢？這就得要藉由『 && 』及『 || 』的幫忙了！ 注意喔，兩個 & 之間是沒有空格的！那個 | 則是 [Shift]+[\] 的按鍵結果。

指令下達情況	說明
cmd1 && cmd2	1. 若 cmd1 執行完畢且正確執行($?=0)，則開始執行 cmd2。 2. 若 cmd1 執行完畢且為錯誤 ($?≠0)，則 cmd2 不執行。
cmd1 || cmd2	1. 若 cmd1 執行完畢且正確執行($?=0)，則 cmd2 不執行。 2. 若 cmd1 執行完畢且為錯誤 ($?≠0)，則開始執行 cmd2。

上述的 cmd1 及 cmd2 都是指令。好了，回到我們剛剛假想的情況，就是想要： (1)先判斷一個目錄是否存在； (2)若存在才在該目錄底下建立一個檔案。由於我們尚未介紹如何判斷式 (test) 的使用，在這裡我們使用 ls 以及回傳值來判斷目錄是否存在啦！ 讓我們進行底下這個練習看看：

```
範例一：使用 ls 查閱目錄 /tmp/abc 是否存在，若存在則用 touch 建立 /tmp/abc/hehe
[root@www ~]# ls /tmp/abc && touch /tmp/abc/hehe
ls: /tmp/abc: No such file or directory
# ls 很乾脆的說明找不到該目錄，但並沒有 touch 的錯誤，表示 touch 並沒有執行

[root@www ~]# mkdir /tmp/abc
[root@www ~]# ls /tmp/abc && touch /tmp/abc/hehe
[root@www ~]# ll /tmp/abc
-rw-r--r-- 1 root root 0 Feb  7 12:43 hehe
```

看到了吧？如果 /tmp/abc 不存在時，touch 就不會被執行，若 /tmp/abc 存在的話，那麼 touch 就會開始執行囉！ 很不錯用吧！不過，我們還得手動自行建立目錄，傷腦筋～能不能自動判斷，如果沒有該目錄就給予建立呢？ 參考一下底下的例子先：

```
範例二：測試 /tmp/abc 是否存在，若不存在則予以建立，若存在就不作任何事情
[root@www ~]# rm -r /tmp/abc                <==先刪除此目錄以方便測試
[root@www ~]# ls /tmp/abc || mkdir /tmp/abc
ls: /tmp/abc: No such file or directory <==真的不存在喔！
[root@www ~]# ll /tmp/abc
total 0                                     <==結果出現了！有進行 mkdir
```

如果你一再重複『 ls /tmp/abc || mkdir /tmp/abc 』畫面也不會出現重複 mkdir 的錯誤！這是因為 /tmp/abc 已經存在， 所以後續的 mkdir 就不會進行！這樣理解否？好了，讓我們再次的討論一下，如果我想要建立 /tmp/abc/hehe 這個檔案， 但我並不知道 /tmp/abc 是否存在，那該如何是好？試看看：

```
範例三：我不清楚 /tmp/abc 是否存在，但就是要建立 /tmp/abc/hehe 檔案
[root@www ~]# ls /tmp/abc || mkdir /tmp/abc && touch /tmp/abc/hehe
```

上面這個範例三總是會建立 /tmp/abc/hehe 的喔！不論 /tmp/abc 是否存在。那麼範例三應該如何解釋呢？ 由於Linux 底下的指令都是由左往右執行的，所以範例三有幾種結果我們來分析一下：

- (1)若 /tmp/abc 不存在故回傳 $?≠0，則 (2)因為 || 遇到非為 0 的 $? 故開始 mkdir /tmp/abc，由於 mkdir /tmp/abc 會成功進行，所以回傳 $?=0 (3)因為 && 遇到 $?=0 故會執行 touch /tmp/abc/hehe，最終 hehe 就被建立了；
- (1)若 /tmp/abc 存在故回傳 $?=0，則 (2)因為 || 遇到 0 的 $? 不會進行，此時 $?=0 繼續向後傳，故 (3)因為 && 遇到 $?=0 就開始建立 /tmp/abc/hehe 了！最終 /tmp/abc/hehe 被建立起來。

整個流程圖示如下：

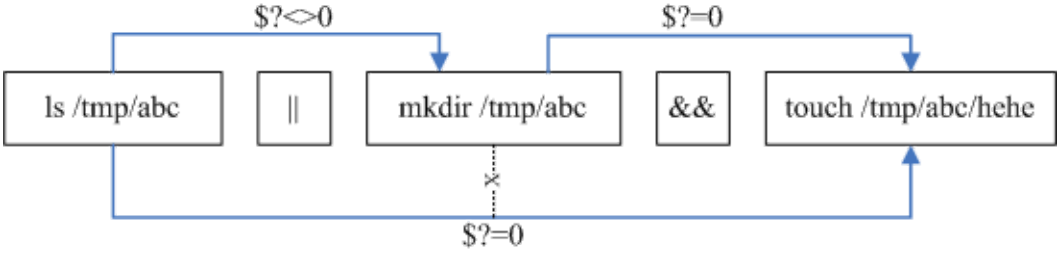

圖 5.2.1、 指令依序執行的關係示意圖

上面這張圖顯示的兩股資料中，上方的線段為不存在 /tmp/abc 時所進行的指令行為，下方的線段則是存在 /tmp/abc 所在的指令行為。如上所述，下方線段由於存在 /tmp/abc 所以導致 $?=0 ，讓中間的 mkdir 就不執行了！ 並將 $?=0 繼續往後傳給後續的 touch 去利用啦！瞭乎？在任何時刻你都可以拿上面這張圖作為示意！ 讓我們來想想底下這個例題吧！

例題：

以 ls 測試 /tmp/vbirding 是否存在，若存在則顯示 "exist" ，若不存在，則顯示 "not exist" ！

答：

這又牽涉到邏輯判斷的問題，如果存在就顯示某個資料，若不存在就顯示其他資料，那我可以這樣做：

```
ls /tmp/vbirding && echo "exist" || echo "not exist"
```

意思是說，當 ls /tmp/vbirding 執行後，若正確，就執行 echo "exist" ，若有問題，就執行 echo "not exist" ！那如果寫成如下的狀況會出現什麼？

```
ls /tmp/vbirding || echo "not exist" && echo "exist"
```

這其實是有問題的，為什麼呢？由圖 5.2.1 的流程介紹我們知道指令是一個一個往後執行， 因此在上面的例子當中，如果 /tmp/vbirding 不存在時，他會進行如下動作：

1. 若 ls /tmp/vbirding 不存在，因此回傳一個非為 0 的數值；
2. 接下來經過 || 的判斷，發現前一個指令回傳非為 0 的數值，因此，程式開始執行 echo "not exist"， 而 echo "not exist" 程式肯定可以執行成功，因此會回傳一個 0 值給後面的指令；
3. 經過 && 的判斷，咦！是 0 啊！所以就開始執行 echo "exist" 。

所以啊，嘿嘿！第二個例子裡面竟然會同時出現 not exist 與 exist 呢！真神奇～

經過這個例題的練習，你應該會瞭解，由於指令是一個接著一個去執行的，因此，如果真要使用判斷， 那麼這個 && 與 || 的順序就不能搞錯。一般來說，假設判斷式有三個，也就是：

```
command1 && command2 || command3
```

而且順序通常不會變，因為一般來說， command2 與 command3 會放置肯定可以執行成功的指令， 因此，依據上面例題的邏輯分析，您就會曉得為何要如此放置囉～這很有用的啦！而且.....考試也很常考～

管線命令（pipe）

就如同前面所說的， bash 命令執行的時候有輸出的資料會出現！ 那麼如果這群資料必需要經過幾道手續之後才能得到我們所想要的格式，應該如何來設定？ 這就牽涉到管線命令的問題了 (pipe)， 管線命令使用的是『 | 』這個界定符號！ 另外，管線命令與『連續下達命令』是不一樣的呦！ 這點底下我們會再說明。底下我們先舉一個例子來說明一下簡單的管線命令。

假設我們想要知道 /etc/ 底下有多少檔案，那麼可以利用 ls /etc 來查閱，不過， 因為 /etc 底下的檔案太多，導致一口氣就將螢幕塞滿了～不知道前面輸出的內容是啥？此時，我們可以透過 less 指令的協助，利用：

```
[root@www ~]# ls -al /etc | less
```

如此一來，使用 ls 指令輸出後的內容，就能夠被 less 讀取，並且利用 less 的功能，我們就能夠前後翻動相關的資訊了！很方便是吧？我們就來瞭解一下這個管線命令『 | 』的用途吧！ 其實這個管線命令『 | 』僅能處理經由前面一個指令傳來的正確資訊，也就是 standard output 的資訊，對於 stdandard error 並沒有直接處理的能力。那麼整體的管線命令可以使用下圖表示：

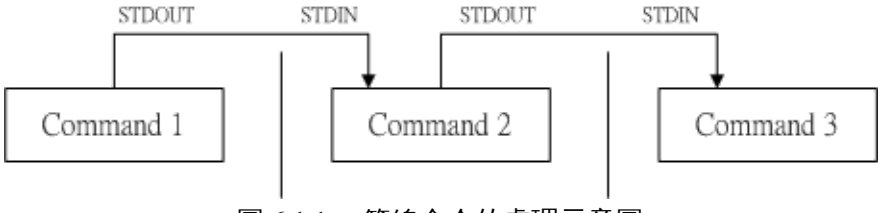

圖 6.1.1、 管線命令的處理示意圖

在每個管線後面接的第一個資料必定是『指令』喔！而且這個指令必須要能夠接受 standard input 的資料才行，這樣的指令才可以是為『管線命令』， 例如 less, more, head, tail 等都是可以接受 standard input 的管線命令啦。至於例如 ls, cp, mv 等就不是管線命令了！因為 ls, cp, mv 並不會接受來自 stdin 的資料。 也就是說， 管線命令主要有兩個比較需要注意的地方：

- 管線命令僅會處理 standard output, 對於 standard error output 會予以忽略

- 管線命令必須要能夠接受來自前一個指令的資料成為 standard input 繼續處理才行。

多說無益，讓我們來玩一些管線命令吧！底下的咚咚對系統管理非常有幫助喔！

擷取命令： cut, grep

什麼是擷取命令啊？說穿了，就是將一段資料經過分析後，取出我們所想要的。或者是經由分析關鍵字，取得我們所想要的那一行！ 不過，要注意的是，一般來說，擷取訊息通常是針對『一行一行』來分析的， 並不是整篇訊息分析的喔～底下我們介紹兩個很常用的訊息擷取命令：

---

cut

cut 不就是『切』嗎？沒錯啦！這個指令可以將一段訊息的某一段給他『切』出來～ 處理的訊息是以『行』為單位喔！底下我們就來談一談：

```
[root@www ~]# cut -d'分隔字元' -f fields <==用於有特定分隔字元
[root@www ~]# cut -c 字元區間            <==用於排列整齊的訊息
選項與參數：
-d  ：後面接分隔字元。與 -f 一起使用；
-f  ：依據 -d 的分隔字元將一段訊息分割成為數段，用 -f 取出第幾段的意思；
-c  ：以字元（characters）的單位取出固定字元區間；

範例一：將 PATH 變數取出，我要找出第五個路徑。
[root@www ~]# echo $PATH
/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/X11R6/bin:/usr/games:
# 1 |    2   |  3  |    4    |       5      |     6        |    7

[root@www ~]# echo $PATH | cut -d ':' -f 5
# 如同上面的數字顯示，我們是以『：』作為分隔，因此會出現 /usr/local/bin
# 那麼如果想要列出第 3 與第 5 呢？，就是這樣：
[root@www ~]# echo $PATH | cut -d ':' -f 3,5

範例二：將 export 輸出的訊息，取得第 12 字元以後的所有字串
[root@www ~]# export
declare -x HISTSIZE="1000"
declare -x INPUTRC="/etc/inputrc"
declare -x KDEDIR="/usr"
declare -x LANG="zh_TW.big5"
.....(其他省略).....
# 注意看，每個資料都是排列整齊的輸出！如果我們不想要『 declare -x 』時，
# 就得這麼做：

[root@www ~]# export | cut -c 12-
HISTSIZE="1000"
INPUTRC="/etc/inputrc"
KDEDIR="/usr"
LANG="zh_TW.big5"
.....(其他省略).....
# 知道怎麼回事了吧？用 -c 可以處理比較具有格式的輸出資料！
# 我們還可以指定某個範圍的值，例如第 12-20 的字元，就是 cut -c 12-20 等等！

範例三：用 last 將顯示的登入者的資訊中，僅留下使用者大名
[root@www ~]# last
root   pts/1    192.168.201.101  Sat Feb  7 12:35   still logged in
root   pts/1    192.168.201.101  Fri Feb  6 12:13 - 18:46  (06:33)
```

```
root   pts/1    192.168.201.254  Thu Feb  5 22:37 - 23:53  (01:16)
# last 可以輸出『帳號/終端機/來源/日期時間』的資料，並且是排列整齊的

[root@www ~]# last | cut -d ' ' -f 1
# 由輸出的結果我們可以發現第一個空白分隔的欄位代表帳號，所以使用如上指令：
# 但是因為 root   pts/1 之間空格有好幾個，並非僅有一個，所以，如果要找出
# pts/1 其實不能以 cut -d ' ' -f 1,2 喔！輸出的結果會不是我們想要的。
```

cut 主要的用途在於將『同一行裡面的資料進行分解！』最常使用在分析一些數據或文字資料的時候！ 這是因為有時候我們會以某些字元當作分割的參數，然後來將資料加以切割，以取得我們所需要的資料。 鳥哥也很常使用這個功能呢！尤其是在分析 log 檔案的時候！不過，cut 在處理多空格相連的資料時，可能會比較吃力一點。

- **grep**

剛剛的 cut 是將一行訊息當中，取出某部分我們想要的，而 grep 則是分析一行訊息， 若當中有我們所需要的資訊，就將該行拿出來～簡單的語法是這樣的：

```
[root@www ~]# grep [-acinv] [--color=auto] '搜尋字串' filename
選項與參數：
-a ：將 binary 檔案以 text 檔案的方式搜尋資料
-c ：計算找到 '搜尋字串' 的次數
-i ：忽略大小寫的不同，所以大小寫視為相同
-n ：順便輸出行號
-v ：反向選擇，亦即顯示出沒有 '搜尋字串' 內容的那一行！
--color=auto ：可以將找到的關鍵字部分加上顏色的顯示喔！

範例一：將 last 當中，有出現 root 的那一行就取出來；
[root@www ~]# last | grep 'root'

範例二：與範例一相反，只要沒有 root 的就取出！
[root@www ~]# last | grep -v 'root'

範例三：在 last 的輸出訊息中，只要有 root 就取出，並且僅取第一欄
[root@www ~]# last | grep 'root' |cut -d ' ' -f1
# 在取出 root 之後，利用上個指令 cut 的處理，就能夠僅取得第一欄囉！

範例四：取出 /etc/man.config 內含 MANPATH 的那幾行
[root@www ~]# grep --color=auto 'MANPATH' /etc/man.config
....(前面省略)....
MANPATH_MAP     /usr/X11R6/bin          /usr/X11R6/man
MANPATH_MAP     /usr/bin/X11            /usr/X11R6/man
MANPATH_MAP     /usr/bin/mh             /usr/share/man
# 神奇的是，如果加上 --color=auto 的選項，找到的關鍵字部分會用特殊顏色顯示喔！
```

grep 是個很棒的指令喔！他支援的語法實在是太多了～用在正規表示法裡頭， 能夠處理的資料實在是多的很～不過，我們這裡先不談正規表示法～下一章再來說明～ 您先瞭解一下， grep 可以解析一行文字，取得關鍵字，若該行有存在關鍵字，就會整行列出來！

排序命令： sort, wc, uniq

很多時候，我們都會去計算一次資料裡頭的相同型態的資料總數，舉例來說， 使用 last 可以查得這個

月份有登入主機者的身份。那麼我可以針對每個使用者查出他們的總登入次數嗎？ 此時就得要排序與計算之類的指令來輔助了！底下我們介紹幾個好用的排序與統計指令喔！

- **sort**

sort 是很有趣的指令，他可以幫我們進行排序，而且可以依據不同的資料型態來排序喔！ 例如數字與文字的排序就不一樣。此外，排序的字元與語系的編碼有關，因此， 如果您需要排序時，建議使用 LANG=C 來讓語系統一，資料排序比較好一些。

```
[root@www ~]# sort [-fbMnrtuk] [file or stdin]
選項與參數：
-f  ：忽略大小寫的差異，例如 A 與 a 視為編碼相同；
-b  ：忽略最前面的空白字元部分；
-M  ：以月份的名字來排序，例如 JAN, DEC 等等的排序方法；
-n  ：使用『純數字』進行排序(預設是以文字型態來排序的)；
-r  ：反向排序；
-u  ：就是 uniq ，相同的資料中，僅出現一行代表；
-t  ：分隔符號，預設是用 [tab] 鍵來分隔；
-k  ：以那個區間 (field) 來進行排序的意思

範例一：個人帳號都記錄在 /etc/passwd 下，請將帳號進行排序。
[root@www ~]# cat /etc/passwd | sort
adm:x:3:4:adm:/var/adm:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
# 鳥哥省略很多的輸出～由上面的資料看起來， sort 是預設『以第一個』資料來排序，
# 而且預設是以『文字』型態來排序的喔！所以由 a 開始排到最後囉！

範例二：/etc/passwd 內容是以 ： 來分隔的，我想以第三欄來排序，該如何？
[root@www ~]# cat /etc/passwd | sort -t ':' -k 3
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
# 看到特殊字體的輸出部分了吧？怎麼會這樣排列啊？呵呵！沒錯啦～
# 如果是以文字型態來排序的話，原本就會是這樣，想要使用數字排序：
# cat /etc/passwd | sort -t ':' -k 3 -n
# 這樣才行啊！用那個 -n 來告知 sort 以數字來排序啊！

範例三：利用 last ，將輸出的資料僅取帳號，並加以排序
[root@www ~]# last | cut -d ' ' -f1 | sort
```

sort 同樣是很常用的指令呢！因為我們常常需要比較一些資訊啦！ 舉個上面的第二個例子來說好了！今天假設你有很多的帳號，而且你想要知道最大的使用者 ID 目前到哪一號了！呵呵！使用 sort 一下子就可以知道答案咯！當然其使用還不止此啦！ 有空的話不妨玩一玩！

- **uniq**

如果我排序完成了，想要將重複的資料僅列出一個顯示，可以怎麼做呢？

```
[root@www ~]# uniq [-ic]
選項與參數：
```

```
-i   ：忽略大小寫字元的不同；
-c   ：進行計數

範例一：使用 last 將帳號列出，僅取出帳號欄，進行排序後僅取出一位；
[root@www ~]# last | cut -d ' ' -f1 | sort | uniq

範例二：承上題，如果我還想要知道每個人的登入總次數呢？
[root@www ~]# last | cut -d ' ' -f1 | sort | uniq -c
      1
     12 reboot
     41 root
      1 wtmp
# 從上面的結果可以發現 reboot 有 12 次， root 登入則有 41 次！
# wtmp 與第一行的空白都是 last 的預設字元，那兩個可以忽略的！
```

這個指令用來將『重複的行刪除掉只顯示一個』，舉個例子來說， 你要知道這個月份登入你主機的使用者有誰，而不在乎他的登入次數，那麼就使用上面的範例， (1)先將所有的資料列出；(2)再將人名獨立出來；(3)經過排序；(4)只顯示一個！ 由於這個指令是在將重複的東西減少，所以當然需要『配合排序過的檔案』來處理囉！

- **wc**

如果我想要知道 /etc/man.config 這個檔案裡面有多少字？多少行？多少字元的話， 可以怎麼做呢？其實可以利用 wc 這個指令來達成喔！他可以幫我們計算輸出的訊息的整體資料！

```
[root@www ~]# wc [-lwm]
選項與參數：
-l   ：僅列出行；
-w   ：僅列出多少字(英文單字)；
-m   ：多少字元；

範例一：那個 /etc/man.config 裡面到底有多少相關字、行、字元數？
[root@www ~]# cat /etc/man.config | wc
    141     722    4617
# 輸出的三個數字中，分別代表： 『行、字數、字元數』

範例二：我知道使用 last 可以輸出登入者，但是 last 最後兩行並非帳號內容，
        那麼請問，我該如何以一行指令串取得這個月份登入系統的總人次？
[root@www ~]# last | grep [a-zA-Z] | grep -v 'wtmp' | wc -l
# 由於 last 會輸出空白行與 wtmp 字樣在最底下兩行，因此，我利用
# grep 取出非空白行，以及去除 wtmp 那一行，在計算行數，就能夠瞭解囉！
```

wc 也可以當作指令？這可不是上洗手間的 WC 呢！這是相當有用的計算檔案內容的一個工具組喔！舉個例子來說， 當你要知道目前你的帳號檔案中有多少個帳號時，就使用這個方法：『 cat /etc/passwd | wc -l 』啦！因為 /etc/passwd 裡頭一行代表一個使用者呀！ 所以知道行數就曉得有多少的帳號在裡頭了！而如果要計算一個檔案裡頭有多少個字元時，就使用 wc -m 這個選項吧！

雙向重導向： tee

想個簡單的東西，我們由前一節知道 > 會將資料流整個傳送給檔案或裝置，因此我們除非去讀取該檔案或裝置， 否則就無法繼續利用這個資料流。萬一我想要將這個資料流的處理過程中將某段訊息存下來，應該怎麼做？ 利用 tee 就可以囉～我們可以這樣簡單的看一下：

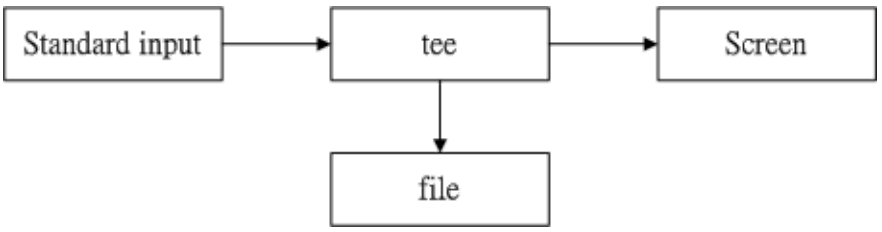

圖 6.3.1、tee 的工作流程示意圖

tee 會同時將資料流分送到檔案去與螢幕 (screen)；而輸出到螢幕的，其實就是 stdout， 可以讓下個指令繼續處理喔！

```
[root@www ~]# tee [-a] file
選項與參數：
-a   ：以累加 (append) 的方式，將資料加入 file 當中！

[root@www ~]# last | tee last.list | cut -d " " -f1
# 這個範例可以讓我們將 last 的輸出存一份到 last.list 檔案中；

[root@www ~]# ls -l /home | tee ~/homefile | more
# 這個範例則是將 ls 的資料存一份到 ~/homefile ， 同時螢幕也有輸出訊息！

[root@www ~]# ls -l / | tee -a ~/homefile | more
# 要注意！ tee 後接的檔案會被覆蓋，若加上 -a 這個選項則能將訊息累加。
```

tee 可以讓 standard output 轉存一份到檔案內並將同樣的資料繼續送到螢幕去處理！ 這樣除了可以讓我們同時分析一份資料並記錄下來之外，還可以作為處理一份資料的中間暫存檔記錄之用！ tee 這傢伙在很多選擇/填充的認證考試中很容易考呢！

字元轉換命令： tr, col, join, paste, expand

我們在 vim 程式編輯器當中，提到過 DOS 斷行字元與 Unix 斷行字元的不同，並且可以使用 dos2unix 與 unix2dos 來完成轉換。好了，那麼思考一下，是否還有其他常用的字元替代？ 舉例來說，要將大寫改成小寫，或者是將資料中的 [tab] 按鍵轉成空白鍵？還有，如何將兩篇訊息整合成一篇？ 底下我們就來介紹一下這些字元轉換命令在管線當中的使用方法：

- **tr**

tr 可以用來刪除一段訊息當中的文字，或者是進行文字訊息的替換！

```
[root@www ~]# tr [-ds] SET1 ...
選項與參數：
-d   ：刪除訊息當中的 SET1 這個字串；
-s   ：取代掉重複的字元！

範例一：將 last 輸出的訊息中，所有的小寫變成大寫字元：
[root@www ~]# last | tr '[a-z]' '[A-Z]'
# 事實上，沒有加上單引號也是可以執行的，如：『 last | tr [a-z] [A-Z] 』

範例二：將 /etc/passwd 輸出的訊息中，將冒號 (:) 刪除
[root@www ~]# cat /etc/passwd | tr -d ':'

範例三：將 /etc/passwd 轉存成 dos 斷行到 /root/passwd 中，再將 ^M 符號刪除
```

```
[root@www ~]# cp /etc/passwd /root/passwd && unix2dos /root/passwd
[root@www ~]# file /etc/passwd /root/passwd
/etc/passwd:  ASCII text
/root/passwd: ASCII text, with CRLF line terminators <==就是 DOS 斷行
[root@www ~]# cat /root/passwd | tr -d '\r' > /root/passwd.linux
# 那個 \r 指的是 DOS 的斷行字元, 關於更多的字符, 請參考 man tr
[root@www ~]# ll /etc/passwd /root/passwd*
-rw-r--r-- 1 root root 1986 Feb  6 17:55 /etc/passwd
-rw-r--r-- 1 root root 2030 Feb  7 15:55 /root/passwd
-rw-r--r-- 1 root root 1986 Feb  7 15:57 /root/passwd.linux
# 處理過後, 發現檔案大小與原本的 /etc/passwd 就一致了 !
```

其實這個指令也可以寫在『正規表示法』裡頭！因為他也是由正規表示法的方式來取代資料的！ 以上面的例子來說，使用 [] 可以設定一串字呢！也常常用來取代檔案中的怪異符號！ 例如上面第三個例子當中，可以去除 DOS 檔案留下來的 ^M 這個斷行的符號！這東西相當的有用！相信處理 Linux & Windows 系統中的人們最麻煩的一件事就是這個事情啦！亦即是 DOS 底下會自動的在每行行尾加入 ^M 這個斷行符號！這個時候我們可以使用這個 tr 來將 ^M 去除！ ^M 可以使用 \r 來代替之！

- **col**

```
[root@www ~]# col [-xb]
選項與參數：
-x   ：將 tab 鍵轉換成對等的空白鍵
-b   ：在文字內有反斜線 (/) 時, 僅保留反斜線最後接的那個字元

範例一：利用 cat -A 顯示出所有特殊按鍵, 最後以 col 將 [tab] 轉成空白
[root@www ~]# cat -A /etc/man.config  <==此時會看到很多 ^I 的符號, 那就是 tab
[root@www ~]# cat /etc/man.config | col -x | cat -A | more
# 嘿嘿！如此一來, [tab] 按鍵會被取代成為空白鍵, 輸出就美觀多了 !

範例二：將 col 的 man page 轉存成為 /root/col.man 的純文字檔
[root@www ~]# man col > /root/col.man
[root@www ~]# vi /root/col.man
COL(1)          BSD General Commands Manual              COL(1)

N^HNA^HAM^HME^HE
     c^Hco^Hol^Hl - filter reverse line feeds from input

S^HSY^HYN^HNO^HOP^HPS^HSI^HIS^HS
     c^Hco^Hol^Hl [-^H-b^Hbf^Hfp^Hpx^Hx] [-^H-l^Hl _^Hn_^Hu_^Hm]
# 你沒看錯！由於 man page 內有些特殊按鈕會用來作為類似特殊按鍵與顏色顯示,
# 所以這個檔案內就會出現如上所示的一堆怪異字元(有 ^ 的)

[root@www ~]# man col | col -b > /root/col.man
```

雖然 col 有他特殊的用途，不過，很多時候，他可以用來簡單的處理將 [tab] 按鍵取代成為空白鍵！ 例如上面的例子當中，如果使用 cat -A 則 [tab] 會以 ^I 來表示。 但經過 col -x 的處理，則會將 [tab] 取代成為對等的空白鍵！此外， col 經常被利用於將 man page 轉存為純文字檔以方便查閱的功能！如上述的範例二！

- **join**

join 看字面上的意義 (加入/參加) 就可以知道，他是在處理兩個檔案之間的資料， 而且，主要是在處理『兩個檔案當中，有 **"相同資料"** 的那一行，才將他加在一起』的意思。我們利用底下的簡單例子

來說明：

```
[root@www ~]# join [-ti12] file1 file2
選項與參數：
-t   ：join 預設以空白字元分隔資料，並且比對『第一個欄位』的資料，
       如果兩個檔案相同，則將兩筆資料聯成一行，且第一個欄位放在第一個！
-i   ：忽略大小寫的差異；
-1   ：這個是數字的 1 ，代表『第一個檔案要用那個欄位來分析』的意思；
-2   ：代表『第二個檔案要用那個欄位來分析』的意思。

範例一：用 root 的身份，將 /etc/passwd 與 /etc/shadow 相關資料整合成一欄
[root@www ~]# head -n 3 /etc/passwd /etc/shadow
==> /etc/passwd <==
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin

==> /etc/shadow <==
root:$1$/3AQpE5e$y9A/D0bh6rElAs:14120:0:99999:7:::
bin:*:14126:0:99999:7:::
daemon:*:14126:0:99999:7:::
# 由輸出的資料可以發現這兩個檔案的最左邊欄位都是帳號！且以 ： 分隔

[root@www ~]# join -t ':' /etc/passwd /etc/shadow
root:x:0:0:root:/root:/bin/bash:$1$/3AQpE5e$y9A/D0bh6rElAs:14120:0:99999:7:::
bin:x:1:1:bin:/bin:/sbin/nologin:*:14126:0:99999:7:::
daemon:x:2:2:daemon:/sbin:/sbin/nologin:*:14126:0:99999:7:::
# 透過上面這個動作，我們可以將兩個檔案第一欄位相同者整合成一行！
# 第二個檔案的相同欄位並不會顯示(因為已經在第一行了嘛！)

範例二：我們知道 /etc/passwd 第四個欄位是 GID ，那個 GID 記錄在
        /etc/group 當中的第三個欄位，請問如何將兩個檔案整合？
[root@www ~]# head -n 3 /etc/passwd /etc/group
==> /etc/passwd <==
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin

==> /etc/group <==
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
# 從上面可以看到，確實有相同的部分喔！趕緊來整合一下！

[root@www ~]# join -t ':' -1 4 /etc/passwd -2 3 /etc/group
0:root:x:0:root:/root:/bin/bash:root:x:root
1:bin:x:1:bin:/bin:/sbin/nologin:bin:x:root,bin,daemon
2:daemon:x:2:daemon:/sbin:/sbin/nologin:daemon:x:root,bin,daemon
# 同樣的，相同的欄位部分被移動到最前面了！所以第二個檔案的內容就沒再顯示。
# 請讀者們配合上述顯示兩個檔案的實際內容來比對！
```

這個 join 在處理兩個相關的資料檔案時，就真的是很有幫助的啦！ 例如上面的案例當中，我的 /etc/passwd, /etc/shadow, /etc/group 都是有相關性的， 其中 /etc/passwd, /etc/shadow 以帳號為相關性，至於 /etc/passwd, /etc/group 則以所謂的 GID (帳號的數字定義) 來作為他的相關性。根據這個相關性，我們可以將有關係的資料放置在一起！這在處理資料可是相當有幫助的！ 但是上面的例子有點難，希望您可以靜下心好好的看一看原因喔！

此外，需要特別注意的是，在使用 join 之前，你所需要處理的檔案應該要事先經過排序（sort）處理！否則有些比對的項目會被略過呢！特別注意了！

- **paste**

這個 paste 就要比 join 簡單多了！相對於 join 必須要比對兩個檔案的資料相關性， paste 就直接『將兩行貼在一起，且中間以 [tab] 鍵隔開』而已！簡單的使用方法：

```
[root@www ~]# paste [-d] file1 file2
選項與參數：
-d   ：後面可以接分隔字元。預設是以 [tab] 來分隔的！
-    ：如果 file 部分寫成 - ，表示來自 standard input 的資料的意思。

範例一：將 /etc/passwd 與 /etc/shadow 同一行貼在一起
[root@www ~]# paste /etc/passwd /etc/shadow
bin:x:1:1:bin:/bin:/sbin/nologin        bin:*:14126:0:99999:7:::
daemon:x:2:2:daemon:/sbin:/sbin/nologin daemon:*:14126:0:99999:7:::
adm:x:3:4:adm:/var/adm:/sbin/nologin    adm:*:14126:0:99999:7:::
# 注意喔！同一行中間是以 [tab] 按鍵隔開的！

範例二：先將 /etc/group 讀出(用 cat)，然後與範例一貼上一起！且僅取出前三行
[root@www ~]# cat /etc/group|paste /etc/passwd /etc/shadow -|head -n 3
# 這個例子的重點在那個 - 的使用！那玩意兒常常代表 stdin 喔！
```

- **expand**

這玩意兒就是在將 [tab] 按鍵轉成空白鍵啦～可以這樣玩：

```
[root@www ~]# expand [-t] file
選項與參數：
-t   ：後面可以接數字。一般來說，一個 tab 按鍵可以用 8 個空白鍵取代。
       我們也可以自行定義一個 [tab] 按鍵代表多少個字元呢！

範例一：將 /etc/man.config 內行首為 MANPATH 的字樣就取出；僅取前三行；
[root@www ~]# grep '^MANPATH' /etc/man.config | head -n 3
MANPATH /usr/man
MANPATH /usr/share/man
MANPATH /usr/local/man
# 行首的代表標誌為 ^ ，這個我們留待下節介紹！先有概念即可！

範例二：承上，如果我想要將所有的符號都列出來？(用 cat)
[root@www ~]# grep '^MANPATH' /etc/man.config | head -n 3 |cat -A
MANPATH^I/usr/man$
MANPATH^I/usr/share/man$
MANPATH^I/usr/local/man$
# 發現差別了嗎？沒錯～ [tab] 按鍵可以被 cat -A 顯示成為 ^I

範例三：承上，我將 [tab] 按鍵設定成 6 個字元的話？
[root@www ~]# grep '^MANPATH' /etc/man.config | head -n 3 | \
>  expand -t 6 - | cat -A
MANPATH     /usr/man$
MANPATH     /usr/share/man$
MANPATH     /usr/local/man$
123456123456123456.....
```

```
# 仔細看一下上面的數字說明，因為我是以 6 個字元來代表一個 [tab] 的長度，所以，
# MAN... 到 /usr 之間會隔 12 (兩個 [tab]) 個字元喔！如果 tab 改成 9 的話，
# 情況就又不同了！這裡也不好理解～您可以多設定幾個數字來查閱就曉得！
```

expand 也是挺好玩的～他會自動將 [tab] 轉成空白鍵～所以，以上面的例子來說， 使用 cat -A 就會查不到 ^I 的字符囉～此外，因為 [tab] 最大的功能就是格式排列整齊！ 我們轉成空白鍵後，這個空白鍵也會依據我們自己的定義來增加大小～ 所以，並不是一個 ^I 就會換成 8 個空白喔！這個地方要特別注意的哩！ 此外，您也可以參考一下 unexpand 這個將空白轉成 [tab] 的指令功能啊！ ^_^

分割命令： split

如果你有檔案太大，導致一些攜帶式裝置無法複製的問題，嘿嘿！找 split 就對了！ 他可以幫你將一個大檔案，依據檔案大小或行數來分割，就可以將大檔案分割成為小檔案了！ 快速又有效啊！真不錯～

```
[root@www ~]# split [-bl] file PREFIX
選項與參數：
-b   ：後面可接欲分割成的檔案大小，可加單位，例如 b, k, m 等；
-l   ：以行數來進行分割。
PREFIX  ：代表前置字元的意思，可作為分割檔案的前導文字。

範例一：我的 /etc/termcap 有七百多K, 若想要分成 300K 一個檔案時？
[root@www ~]# cd /tmp; split -b 300k /etc/termcap termcap
[root@www tmp]# ll -k termcap*
-rw-r--r-- 1 root root 300 Feb  7 16:39 termcapaa
-rw-r--r-- 1 root root 300 Feb  7 16:39 termcapab
-rw-r--r-- 1 root root 189 Feb  7 16:39 termcapac
# 那個檔名可以隨意取的啦！我們只要寫上前導文字，小檔案就會以
# xxxaa, xxxab, xxxac 等方式來建立小檔案的！

範例二：如何將上面的三個小檔案合成一個檔案，檔名為 termcapback
[root@www tmp]# cat termcap* >> termcapback
# 很簡單吧？就用資料流重導向就好啦！簡單！

範例三：使用 ls -al / 輸出的資訊中，每十行記錄成一個檔案
[root@www tmp]# ls -al / | split -l 10 - lsroot
[root@www tmp]# wc -l lsroot*
  10 lsrootaa
  10 lsrootab
   6 lsrootac
  26 total
# 重點在那個 - 啦！一般來說，如果需要 stdout/stdin 時，但偏偏又沒有檔案，
# 有的只是 - 時，那麼那個 - 就會被當成 stdin 或 stdout ～
```

在 Windows 作業系統下，你要將檔案分割需要如何作？傷腦筋吧！在 Linux 底下就簡單的多了！你要將檔案分割的話，那麼就使用 -b size 來將一個分割的檔案限制其大小，如果是行數的話，那麼就使用 -l line 來分割！好用的很！如此一來，你就可以輕易的將你的檔案分割成軟碟 (floppy) 的大小，方便你 copy 囉！

參數代換： xargs

xargs 是在做什麼的呢？就以字面上的意義來看， x 是加減乘除的乘號，args 則是 arguments (參數) 的

意思，所以說，這個玩意兒就是在產生某個指令的參數的意思！ xargs 可以讀入 stdin 的資料，並且以空白字元或斷行字元作為分辨，將 stdin 的資料分隔成為 arguments 。 因為是以空白字元作為分隔，所以，如果有一些檔名或者是其他意義的名詞內含有空白字元的時候， xargs 可能就會誤判了～他的用法其實也還滿簡單的！就來看一看先！

```
[root@www ~]# xargs [-0epn] command
選項與參數：
-0   ：如果輸入的 stdin 含有特殊字元，例如 `, \, 空白鍵等等字元時，這個 -0 參數
       可以將他還原成一般字元。這個參數可以用於特殊狀態喔！
-e   ：這個是 EOF (end of file) 的意思。後面可以接一個字串，當 xargs 分析到
       這個字串時，就會停止繼續工作！
-p   ：在執行每個指令的 argument 時，都會詢問使用者的意思；
-n   ：後面接次數，每次 command 指令執行時，要使用幾個參數的意思。看範例三。
當 xargs 後面沒有接任何的指令時，預設是以 echo 來進行輸出喔！

範例一：將 /etc/passwd 內的第一欄取出，僅取三行，使用 finger 這個指令將每個
        帳號內容秀出來
[root@www ~]# cut -d':' -f1 /etc/passwd |head -n 3| xargs finger
Login: root                             Name: root
Directory: /root                        Shell: /bin/bash
Never logged in.
No mail.
No Plan.
......底下省略.....
# 由 finger account 可以取得該帳號的相關說明內容，例如上面的輸出就是 finger root
# 後的結果。在這個例子當中，我們利用 cut 取出帳號名稱，用 head 取出三個帳號，
# 最後則是由 xargs 將三個帳號的名稱變成 finger 後面需要的參數！

範例二：同上，但是每次執行 finger 時，都要詢問使用者是否動作？
[root@www ~]# cut -d':' -f1 /etc/passwd |head -n 3| xargs -p finger
finger root bin daemon ?...y
.....(底下省略)....
# 呵呵！這個 -p 的選項可以讓使用者的使用過程中，被詢問到每個指令是否執行！

範例三：將所有的 /etc/passwd 內的帳號都以 finger 查閱，但一次僅查閱五個帳號
[root@www ~]# cut -d':' -f1 /etc/passwd | xargs -p -n 5 finger
finger root bin daemon adm lp ?...y
.....(中間省略)....
finger uucp operator games gopher ftp ?...y
.....(底下省略)....
# 在這裡鳥哥使用了 -p 這個參數來讓您對於 -n 更有概念。一般來說，某些指令後面
# 可以接的 arguments 是有限制的，不能無限制的累加，此時，我們可以利用 -n
# 來幫助我們將參數分成數個部分，每個部分分別再以指令來執行！這樣就 OK 啦！^_^

範例四：同上，但是當分析到 lp 就結束這串指令？
[root@www ~]# cut -d':' -f1 /etc/passwd | xargs -p -e'lp' finger
finger root bin daemon adm ?...
# 仔細與上面的案例做比較。也同時注意，那個 -e'lp' 是連在一起的，中間沒有空白鍵。
# 上個例子當中，第五個參數是 lp 啊，那麼我們下達 -e'lp' 後，則分析到 lp
# 這個字串時，後面的其他 stdin 的內容就會被 xargs 捨棄掉了！
```

其實，在 man xargs 裡面就有三四個小範例，您可以自行參考一下內容。 此外， xargs 真的是很好用的一個玩意兒！您真的需要好好的參詳參詳！會使用 xargs 的原因是， 很多指令其實並不支援管線命令，因此我們可以透過 xargs 來提供該指令引用 standard input 之用！舉例來說，我們使用如下的範例來說明：

```
範例五：找出 /sbin 底下具有特殊權限的檔名，並使用 ls -l 列出詳細屬性
[root@www ~]# find /sbin -perm +7000 | ls -l
# 結果竟然僅有列出 root 所在目錄下的檔案！這不是我們要的！
# 因為 ll (ls) 並不是管線命令的原因啊！

[root@www ~]# find /sbin -perm +7000 | xargs ls -l
-rwsr-xr-x 1 root root 70420 May 25  2008 /sbin/mount.nfs
-rwsr-xr-x 1 root root 70424 May 25  2008 /sbin/mount.nfs4
-rwxr-sr-x 1 root root  5920 Jun 15  2008 /sbin/netreport
....(底下省略)....
```

關於減號 - 的用途

管線命令在 bash 的連續的處理程序中是相當重要的！另外，在 log file 的分析當中也是相當重要的一環， 所以請特別留意！另外，在管線命令當中，常常會使用到前一個指令的 stdout 作為這次的 stdin ， 某些指令需要用到檔案名稱 (例如 tar) 來進行處理時，該 stdin 與 stdout 可以利用減號 "-" 來替代，舉例來說：

```
[root@www ~]# tar -cvf - /home | tar -xvf -
```

上面這個例子是說：『我將 /home 裡面的檔案給他打包，但打包的資料不是紀錄到檔案，而是傳送到 stdout； 經過管線後，將 tar -cvf - /home 傳送給後面的 tar -xvf - 』。後面的這個 - 則是取用前一個指令的 stdout， 因此，我們就不需要使用 file 了！這是很常見的例子喔！注意注意！

重點回顧

- 由於核心在記憶體中是受保護的區塊，因此我們必須要透過『 Shell 』將我們輸入的指令與 Kernel 溝通，好讓 Kernel 可以控制硬體來正確無誤的工作
- 學習 shell 的原因主要有：文字介面的 shell 在各大 distribution 都一樣；遠端管理時文字介面速度較快； shell 是管理 Linux 系統非常重要的一環，因為 Linux 內很多控制都是以 shell 撰寫的。
- 系統合法的 shell 均寫在 /etc/shells 檔案中；
- 使用者預設登入取得的 shell 記錄於 /etc/passwd 的最後一個欄位；
- bash 的功能主要有：命令編修能力；命令與檔案補全功能；命令別名設定功能；工作控制、前景背景控制；程式化腳本；萬用字元
- type 可以用來找到執行指令為何種類型，亦可用於與 which 相同的功能；
- 變數就是以一組文字或符號等，來取代一些設定或者是一串保留的資料
- 變數主要有環境變數與自訂變數，或稱為全域變數與區域變數
- 使用 env 與 export 可觀察環境變數，其中 export 可以將自訂變數轉成環境變數；
- set 可以觀察目前 bash 環境下的所有變數；
- $? 亦為變數，是前一個指令執行完畢後的回傳值。在 Linux 回傳值為 0 代表執行成功；
- locale 可用於觀察語系資料；
- 可用 read 讓使用者由鍵盤輸入變數的值
- ulimit 可用以限制使用者使用系統的資源情況
- bash 的設定檔主要分為 login shell 與 non-login shell。login shell 主要讀取 /etc/profile 與 ~/.bash_profile， non-login shell 則僅讀取 ~/.bashrc
- 萬用字元主要有： *, ?, [] 等等
- 資料流重導向透過 >, 2>, < 之類的符號將輸出的資訊轉到其他檔案或裝置去；
- 連續命令的下達可透過 ; && || 等符號來處理
- 管線命令的重點是：『管線命令僅會處理 standard output，對於 standard error output 會予

以忽略』『管線命令必須要能夠接受來自前一個指令的資料成為 standard input 繼續處理才行。』
- 本章介紹的管線命令主要有：cut, grep, sort, wc, uniq, tee, tr, col, join, paste, expand, split, xargs 等。

本章習題

(要看答案請將滑鼠移動到『答：』底下的空白處，按下左鍵圈選空白處即可察看)

- 情境模擬題一：由於 ~/.bash_history 僅能記錄指令，我想要在每次登出時都記錄時間，並將後續的指令 50 筆記錄下來， 可以如何處理？

  - 目標：瞭解 history，並透過資料流重導向的方式記錄歷史命令；
  - 前提：需要瞭解本章的資料流重導向，以及瞭解 bash 的各個環境設定檔資訊。

  其實處理的方式非常簡單，我們可以瞭解 date 可以輸出時間，而利用 ~/.myhistory 來記錄所有歷史記錄， 而目前最新的 50 筆歷史記錄可以使用 history 50 來顯示，故可以修改 ~/.bash_logout 成為底下的模樣：

```
[root@www ~]# vim ~/.bash_logout
date >> ~/.myhistory
history 50 > > ~/.myhistory
clear
```

簡答題部分：

- 在 Linux 上可以找到哪些 shell(舉出三個) ？那個檔案記錄可用的 shell ？而 Linux 預設的 shell 是？

  1) /bin/bash, /bin/tcsh, /bin/csh
  2) /etc/shells
  3) bash , 亦即是 /bin/bash。

- 在 shell 環境下，有個提示字元 (prompt)，他可以修改嗎？要改什麼？預設的提示字元內容是？

  可以修改的，改 PS1 這個變數，這個 PS1 變數的預設內容為：『[\u@\h \W]\$』

- 如何顯示 HOME 這個環境變數？

  echo $HOME

- 如何得知目前的所有變數與環境變數的設定值？

  環境變數用 env 或 export 而所有變數用 set 即可顯示

- 我是否可以設定一個變數名稱為 3myhome ？

  不行！變數不能以數字做為開頭，參考變數設定規則的內容

- 在這樣的練習中『A=B』且『B=C』，若我下達『unset $A』，則取消的變數是 A 還是 B？

  被取消的是 B 喔，因為 unset $A 相當於 unset B 所以取消的是 B , A 會繼續存在！

- 如何取消變數與命令別名的內容？

  使用 unset 及 unalias 即可

- 如何設定一個變數名稱為 name 內容為 It's my name ？

  name=It\'s\ my\ name 或 name="It's my name"

- bash 環境設定檔主要分為哪兩種類型的讀取？分別讀取哪些重要檔案？

(1)login shell：主要讀取 /etc/profile 及 ~/.bash_profile
(2)non-logni shell：主要讀取 ~/.bashrc 而已。

- CentOS 5.x 的 man page 的路徑設定檔案？

  /etc/man.config

- 試說明 ', ", 與 ` 這些符號在變數定義中的用途？

  參考變數規則那一章節，其中， " 可以具有變數的內容屬性，' 則僅有一般字元，至於 ` 之內則是可先被執行的指令。

- 跳脫符號 \ 有什麼用途？

  可以用來跳脫特殊字元，例如 Enter，$ 等等，使成為一般字元！

- 連續命令中， ;, &&, || 有何不同？

  分號可以讓兩個 command 連續運作，不考慮 command1 的輸出狀態， && 則前一個指令必需要沒有錯誤訊息，亦即回傳值需為 0 則 command2 才會被執行， || 則與 && 相反！

- 如何將 last 的結果中，獨立出帳號，並且印出曾經登入過的帳號？

```
last | cut -d ' ' -f1 | sort | uniq
```

- 請問 foo1 && foo2 | foo3 > foo4，這個指令串當中， foo1/foo2/foo3/foo4 是指令還是檔案？ 整串指令的意義為？

  foo1, foo2 與 foo3 都是指令， foo4 是裝置或檔案。整串指令意義為：
  (1)當 foo1 執行結果有錯誤時，則該指令串結束；
  (2)若 foo1 執行結果沒有錯誤時，則執行 foo2 | foo3 > foo4 ；其中：
  (2-1)foo2 將 stdout 輸出的結果傳給 foo3 處理；
  (2-2)foo3 將來自 foo2 的 stdout 當成 stdin ，處理完後將資料流重新導向 foo4 這個裝置/檔案

- 如何秀出在 /bin 底下任何以 a 為開頭的檔案檔名的詳細資料？

```
ls -l /bin/a*
```

- 如何秀出 /bin 底下，檔名為四個字元的檔案？

```
ls -l /bin/????
```

- 如何秀出 /bin 底下，檔名開頭不是 a-d 的檔案？

```
ls -l /bin/[^a-d]*
```

- 我想要讓終端機介面的登入提示字元修改成我自己喜好的模樣，應該要改哪裡？(filename)

  /etc/issue

- 承上題，如果我是想要讓使用者登入後，才顯示歡迎訊息，又應該要改哪裡？

  /etc/motd

參考資料與延伸閱讀

- 註1：Webmin 的官方網站：http://www.webmin.com/
- 註2：關於 shell 的相關歷史可以參考網路農夫兄所整理的優秀文章。不過由於網路農夫兄所建置的網站暫時關閉， 因此底下的連結為鳥哥到網路上找到的片段文章連結。若有任何侵權事宜，請來信告知，謝謝：
  http://linux.vbird.org/linux_basic/0320bash/csh/
- 註3：使用 man bash，再以 PS1 為關鍵字去查詢，按下數次 n 往後查詢後，可以得到 PS1 的變數說明。
- 註4：i18n 是由一些 Linux distribution 貢獻者共同發起的大型計畫，目的在於讓眾多的 Linux distributions 能夠有良好的萬國碼 (Unicode) 語系的支援。詳細的資料可以參考：

i18n 的官方網站：http://www.openi18n.org/
康橋大學 Dr Markus Kuhn 的文獻：http://www.cl.cam.ac.uk/~mgk25/unicode.html
Debian 社群所寫的文件：http://www.debian.org/doc/manuals/intro-i18n/

- 臥龍小三的教學文件：http://linux.tnc.edu.tw/techdoc/shell/book1.html
- GNU 計畫的 BASH 說明：http://www.gnu.org/manual/bash-2.05a/html_mono/bashref.html
  鳥哥的備份：http://linux.vbird.org/linux_basic/0320bash/0320bash_reference.php
- man bash

2002/06/27：第一次完成
2003/02/10：重新編排與加入 FAQ
2005/08/17：將舊的資料放置到 這裡
2005/08/17：終於稍微搞定了～花了半個多月不眠不休～呼～補充了較多的管線命令與資料流重導向！
2005/08/18：加入額外的變數設定部分！
2005/08/30：加入了 login 與 non-login shell 的簡單說明！
2006/03/19：原先在 col 的說明當中，原本指令『cat -A /etc/man.config | col -x | cat -A | more』不該有 -A！
2006/10/05：感謝小州兄的告知，修正了原本 ~/.bashrc 說明當中的錯誤。
2007/04/05：原本的 cut 範例說明有誤，原本是『我要找出第三個』應該改為『我要找出第五個』才對！
2007/04/11：原本的 join 說明沒有加上排序，應該需要排序後再處理才對！
2007/07/15：原本的額外的變數功能表格有誤，在 var=${str+expr} 與var=${str:+expr} 需要修改，請參考 此處
2009/01/13：將原本基於 FC4 寫作的舊文章移動到此處
2009/02/03：拿掉了原本的『變數的用途』部分，改以案例說明
2009/02/05：多加了變數刪除、取代與替代部分的範例，看起來應該不會像前一版那樣不容易理解！
2009/08/25：加入了情境模擬，並且進行一些說明的細部修改而已。
2010/04/16：感謝wenyenyang兄的告知， wc -c 錯誤，是 wc -m 才是！
