

# Shell 設計入門

臥龍小三 [ols3@www.tnc.edu.tw](mailto:ols3@www.tnc.edu.tw)

台南縣教育網路中心

[Copyright](#) © 2002 by OLS3

v1.0.0 08/01/2002

v1.0.1 08/05/2002

v1.0.2 08/08/2002

v1.0.3 08/12/2002

v1.0.4 08/15/2002

v1.0.5 08/19/2002

---

## Table of Contents

### [1.前言](#)

### [2.佈置練習環境](#)

[Linux/FreeBSD等 Un\\*x 平台](#)

[Cygwin 環境](#)

### [3.Bash shell 的結構](#)

[簡單的示範程式](#)

[打開執行權](#)

[執行 script 的方法](#)

[shell 使工作自動化容易](#)

[您一定可以學會它](#)

[本節習題](#)

### [4.基本的命令](#)

[bash shell 的內建命令](#)

[echo](#)

[cd](#)

[pwd](#)

[alias](#)

[命令列程式](#)

[date](#)

[who](#)

[ls](#)

[cat](#)

[wc](#)

[ln](#)

[basename](#)

[dirname](#)

[sort](#)

[uniq](#)

[cut](#)

[paste](#)

[tr](#)

[grep](#)

[練習用的資料檔](#)

[常用的特殊字元](#)

[一個命令列執行好幾個命令](#)

[命令列郵寄帶檔的方法](#)

[本節習題](#)

## [5.設定變數](#)

[設定變數的方法](#)

[取得變數的內容](#)

[取消變數的內容](#)

[變數的有效範圍](#)

[使變數成為環境變數](#)

[取消環境變數](#)

[由標準輸入讀取資料](#)

[陣列 \(array\)](#)

[Here Document](#)

[進階變數設定](#)

[亂數產生](#)

[eval: 使 shell 掃描命令二次](#)

[本節習題](#)

## [6.萬用字元](#)

## [7.標準輸入/輸出/錯誤 及I/O轉向 與 管線](#)

[本節習題](#)

## [8.正規表示式](#)

[何謂正規表示式\(Regular Expressions\)](#)

[一點 .](#)

[^](#)

[\\_](#)

[\\$](#)

[\[...\]](#)

[\\*](#)

[-](#)

[\{...\}](#)

[\\(...\\)](#)

[sed](#)

[sed 的作用格式](#)

[sed 的作用法 1](#)

[sed 的作用法 2](#)

[sed 的作用法 3](#)

[sed 的作用法 4](#)

[sed 的作用法 5](#)

[sed 的作用法 6](#)

[sed 的作用法 7](#)

[awk](#)

[awk 的作用格式](#)

[awk 的作用法 1](#)

[awk 的作用法 2](#)

[awk 的作用法 3](#)

[awk 的作用法 4](#)

[awk 的作用法 5](#)

[本節習題](#)

[9.引號](#)

[10.算術運算](#)

[11.參數傳遞](#)

[12.程式條件控制 if 語法結構](#)

[13.真假值判斷](#)

[14.case 語法結構](#)

[本節習題](#)

[15.迴圈 語法結構](#)

[for 的迴圈](#)

[while 的迴圈](#)

[until 的迴圈](#)

[無窮迴圈](#)

[本節習題](#)

[16.函式](#)

[函式的寫法](#)

[引入函式檔](#)

[傳遞參數](#)

[函式也可以遞迴呼叫](#)

[17.select 選單語法](#)

[select 的寫法](#)

[本節習題](#)

[18.歷史記錄](#)

[取用歷史記錄的方法](#)

[取用歷史記錄的最後一筆](#)

[取用最後一筆歷史記錄中的參數部份](#)

[實例](#)

[參考資源](#)

# 版權宣告：1999 Copyright OLS3 All rights reserved.

# 作者：OLS3 (臥龍小三)

# 本講義僅供台南縣 87 學年度網路管理進階研習班上課之用.

# 學員可保存一份自用，供日後網管時參考備查.

# 作者保有一切形式的著作權.

- # 欲作其它用途者，需經作者授權同意.
- # 未經作者授權同意之前，請勿轉載刊登.

# Bash 的環境設定

和 Bash 的環境設定有關的檔案有

/etc/profile (主要)

\$HOME/.bash\_profile (主要)

\$HOME/.bash\_login

\$HOME/.profile

\$HOME/.bash\_logout (主要)

\$HOME/.bashrc (主要)

/etc/bashrc

說明如下：

- 登入(login)時
  1. 先執行 /etc/profile
  2. 接著 bash 會檢查使用者的自家目錄中，是否有 .bash\_profile 或者 .bash\_login 或者 .profile，若有，則會執行其中一個，執行順序為：

- a. `.bash_profile` 最優先
- b. `.bash_login` 其次
- c. `.profile` 最後

這三個檔案只有在登入時，才會被 `bash` 讀取

- 登出(`exit/logout`)時

`bash` 會檢查使用者自家目錄中是否有 `.bash_logout`，若有，則 `bash` 會執行其中的指令

- 登入後啟動一個新的 `shell`：

此時我們稱之為一個 `subshell`，也就是說在命令列中鍵入 `bash`，除了原先登入時的 `bash` 之外，又另外啟動了一個新的 `bash shell`。

`bash` 會檢查使用者的自家目錄中是否有 `.bashrc`，若有則予以執行

- 實驗：

在各檔中加入 `echo` 指令，以觀察其執行順序。

## 各檔案用途說明

- `/etc/profile` 由 `root` 所控管，用來設定適合全體使用者的 `shell` 環境
- 若使用者自己覺得 `/etc/profile` 的設定，並不合意，可以修改自家目錄中的 `.bash_profile`
- 既然有了 `.bash_profile`，為何要有 `.bash_login` 及 `.profile`？這是因為有些人可能是從 `Bourne shell` 移轉過來的，那麼，只要將 `Bourne shell` 主要的啟動檔 `.profile` 移到自家目錄中，放棄使用 `.bash_profile` 及 `.bash_login` 即可繼續沿用以前的設定環境
- `.bash_login` 存在理由或許和 `c shell` 有關，但因為 `bash` 和 `c shell` 二者語法並不完全相容，因此，並不建議將 `c shell` 的啟動檔直接移過來使用。

- .bashrc 則是用來設定 subshell 的環境的，之所以要有這個 .bashrc 是為免 subshell 產生時，又重複將 /etc/profile 執行一次。我們發現 .bashrc 中已預先會去執行 /etc/bashrc 的指令，這表示，或許 root 會將產生 subshell 時的環境設好了，使用者只要沿用 /etc/bashrc 的內容，應該不會有任何問題。
- .bash\_logout 是使用者登出主機之前，會去執行的設定檔，如果使用者希望在他登出系統之後，能幫他自動處理一些瑣事，比如：清除暫存檔，清除螢幕等，可以在這個檔案中加以設定。

# 版權宣告：1999 Copyright OLS3 All rights reserved.

# 作者：OLS3 (臥龍小三)

# 本講義僅供台南縣 87 學年度網路管理進階研習班上課之用。

# 學員可保存一份自用，供日後網管時參考備查。

# 作者保有一切形式的著作權。

# 欲作其它用途者，需經作者授權同意。

# 未經作者授權同意之前，請勿轉載刊登。

# Shell

當我們登入 Linux 之後，第一個接觸到的，便是 Shell。我們必須對它有點初步的認識才行。

## 一. Shell 簡介.

Linux 系統分成三個重要的部份

- 核心
- Shell
- 工具程式

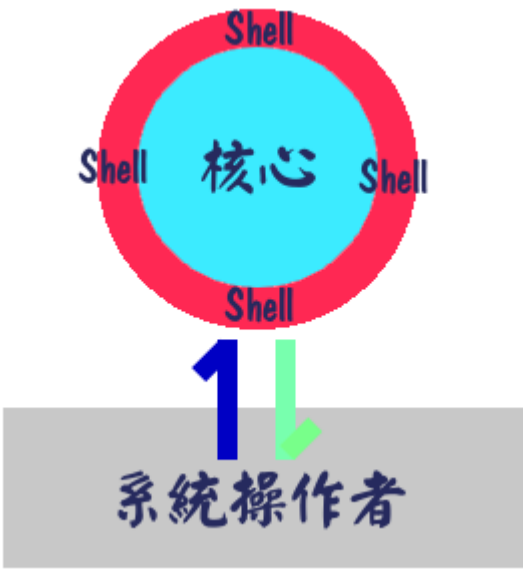
核心的部份相當低階，操作者不易和它直接溝通，因此，必須要有一個友善的介面(interface)，使得操作時能更為方便，這個介面便是 Shell。

換言之，Shell 就是一個居於核心和操作者之間的一層使用者介面。

那麼，為何稱它為 Shell 呢？Shell 的本意是“殼”的意思呢！

沒錯，在核心的外面，包覆著一層外殼，用來負責接收使用者輸入的指令，然後將指令解譯成核心能夠了解的方式，傳給核心去執行，再將結果傳回至預設的輸出周邊。

如圖所示：



例如：鍵入

```
ls -l
```

shell 給你以下回應：

檔案種類	檔案權限	擁有人	組別	檔案大小 (byte 單位)	最近修改的日期及時間	檔案名稱
drwxr-xr-x	14	root	root	1024	Jul 21 21:31	.
drwxr-xr-x	17	root	root	1024	Apr 11 12:01	..
drwxr-xr-x	8	82	82	1024	Feb 25 19:23	apache
-rw-r--r--	1	root	root	1335460	Feb 25 17:45	apache_1_3_4.tar.gz
drwxr-xr-x	6	root	root	1024	Aug 31 1998	ftp
drwxr-xr-x	5	root	root	1024	Aug 31 1998	httpd
drwx-----	2	james	james	1024	Apr 23 06:47	james
drwxr-xr-x	2	root	root	12288	Aug 31 1998	lost+found
drwxr-xr-x	2	msql	nobody	1024	Dec 28 1998	msql
drwxr-xr-x	28	ols3	ols3	2048	Jul 19 21:38	ols3
drwxr-xr-x	3	1022	nobody	1024	Jan 25 23:40	ols3cgi
drwxrwxr-x	3	perl	ols3	1024	Feb 2 07:11	perl
-rw-----	1	root	root	2097152	Jul 21 18:39	quota.group
-rw-----	1	root	root	1278656	Jul 21 18:39	quota.user
drwxrwxr-x	2	root	nobody	1024	May 11 1998	samba
drwxr-xr-x	3	apache	nobody	1024	Feb 25 16:58	temp

```
drwxrwxr-x  2 webadm  nobody      1024 Mar 14 16:34 webadm
```

其實不只是 Linux 有這一層 Shell，其它作業系統也有。  
比如 DOS 的 command.com, Windows 的 GUI(Graphical User Interface), Mac 的 GUI.

Shell 按著表現的方式與讀取使用者輸入種類的不同，可分為二大類：

- Text base : 文字導向
- Graph base: 圖型導向

所謂“讀取使用者輸入種類不同”是指：讀取自鍵盤，或讀取自滑鼠，其它 serial input, 螢幕觸控等.

這樣說來，Shell 好像只是命令直譯器罷了?!  
嗯，這倒要按不同的 OS 所附給的 Shell 其功能和選擇性的自由度而定.

以 DOS 的 COMMAND.COM 而言，它就是一個十足的命令直譯器，除了一點點 batch 檔的能力之外，它的功能並不多。Win 平台的 GUI，則是一個圖型式的命令直譯器，介面十分友善。不過，這二種 OS，不能讓你自由而簡單地選擇 Shell。(以前 DOS 有 4dos 可選用)

Linux 的 shell，除了做為命令直譯器之外，它也是一個不錯的程式語言，是系統管理維護時的重要工具.

由於 Unix 家族，對 Shell 的處理，採獨立自由開放的方式，因此，Shell 的種類相當地多，更可以讓人自由地更換(chsh).

目前流行的 shell 有：

- Bourne shell : sh
- C shell : csh
- Korn shell: ksh (商業軟體)
- tcsh (free)
- Bourne Again shell: bash (GNU)

Linux 的標準 shell 是採用 bash. 它也是我們要學習的主要對象.

## 二. Shell 的簡史

第一個重要的 shell 是 Bourne shell (如此命名是為了紀念此 shell 的發明者 Steven Bourne), 1979 年第一個流行的 Unix 版本 7 發行時，開始使用 Bourne shell.



Bourne shell 的主檔名為 sh，因此，日後人們便以 sh 為 Bourne shell 的主要識別名稱。

雖然 Unix 上的 shell 有許多種，但 Bourne shell 的地位至今仍然沒有改變。許多 Unix 系統中仍然使用 sh 做為重要的管理工具。（它的工作從開機到關機，幾乎無所不包）

第一個廣為流行使用的 shell 變種是 C shell。C shell 主要附在 BSD 版的 Unix 系統中。它的作者是柏克萊大學的 Bill Joy。C shell 主要是因為其語法和 C 語言相類似，因而得名。這使得 Unix 系統的程式師，在學習 C shell 時，感到相當地方便容易。

以上這二種形成 shell 的二十大主流，後來的變種 shell 大都攫取這二種 shell 的優點。

比如 Korn, tcsh 及 bash。

Bash shell 是 GNU 計劃的重要工具軟體之一，也是 GNU 作業系統中標準的 shell。

Bash 相容於 sh，因此，許多早期開發出來的 Bourne shell 都可以繼續在 bash 中運作。現在我們安裝好的 RedHat Linux 便是完全使用 Bash。（/bin/sh -> /bin/bash）

Bash 在 1988 年誕生，最初的作者是 Brian Fox，Chet Ramey 於 1989 加入，現在官方正式的維護者是 Chet Ramey，他的工作便是持續不斷地增強 bash 的功能。

1995~1996 期間推出 bash 2.0，在這之前，廣為使用的版本是 1.14.x，它增加了許多新的功能，以及更好的相容性。

當然，Bash 是完全免費的，它是 Open Source 的一員，原始碼全部開放。

## 二. Bash 的功能.

Bash 具有以下功能：

- 相容於 Bourne shell (sh)
- 包含有 C shell 以及 Korn shell 中最好的功能。
- 具命令列編修的能力(您記得以前 DOS 中的 doskey 嗎?)
- 工作控制(job control)的能力，可控制前景及背景程式
- 具 shell 程式設計的能力，可讓您自訂 shell 及設計程式，管理系統。

## 三. 新版的 Bash 哪裡抓取?

若欲抓取新版的 bash, 可至 <http://www.gnu.org> 或其 mirror 站台.

中研院 FTP 也是不錯的選擇. <ftp://ftp.sinica.edu.tw> 或 <ftp://linux.sinica.edu.tw>

### 三. 開始使用 Bash

當你 login 進 Linux 主機時, 便開始和 bash 互動, 一直到你 logout 主機 (下 exit, logout, 或按 ^D) 為止.

Bash 的提示符號為 \$ (代表一般身份使用者), 當您具有 root 權限時, 提示符號則變為 #.

一旦出現提示符號時, 您便可以開始鍵入操作命令列(command line)了.

命令可分為二大類:

- bash 內建的指令
- 程式

如果是 bash 內建的指令, 則由 bash shell 負責回應; 若是程式, 則 shell 會找出該程式, 然後將控制權交給核心, 由核心執行該程式, 執行完之後, 再將控制權交回給 shell.

怎麼知道那些指令是 bash 內建的, 那些是程式呢? 通常用 "which 指令", 若沒有任何回應, 表示是內建的指令 (除非該指令錯誤、不存在, 或該程式不在預設的搜尋路徑之內), 範例如下:

```
[ols3@ols3 /ols3]$ which echo          [沒有回應, 表示是內建的指令]
```

```
[ols3@ols3 /ols3]$ which ls
```

```
/bin/ls
```

### 四. 命令列的格式.

命令列通常由好幾個字串組成, 中間用空白或 tab 鍵分開. 如下所示:

command options arguments(或稱為 parameters)

命令 選項 參數

rm -rf /home/ols3

除了空白和 tab 鍵之外，每一部份，我們稱之為 token，比如上面的例子中，便有三個 token: rm, -rf, /home/ols3.

當鍵入此一命令列時，shell 首先將它分解成個別的 token，然後判斷是內建的指令，或是程式，再按之前提過的方式去執行。

怎麼知道一個命令或程式，它有那些選項和參數呢？通常 man 一下該指令，就可以得到了。例如：

```
man rm
```

另外，多行指令也可以一下全部寫在同一命令列中，只要中間用 ; 分開，如：

```
ls ; mkdir test ; clear
```

## 五. 現行目錄和自家目錄.

所謂現行目錄(current directory)是指：你現在所處的位置，又稱為工作目錄(working directory).

欲知現行目錄為何？可下 pwd 指令便知。

所謂自家目錄(home directory)是指：當初 root 為你建立帳號時，所指定給你的一個私人專用的目錄，也是你登入系統之後，第一個進入的地方。

欲知自家目錄，可用下列方式：

- cd (然後直接按 Enter)
- cd ~ (~ 代表自家目錄)
- cat /etc/passwd | grep 您的帳號

相關的技巧

- cd ./myway (進入目前目錄下的 myway 目錄中)
- cd .. (回到上一層目錄)
- cd - (回到先前的目錄)

## 六. 萬用字元.

如果命令列的參數中，含有檔名，那麼萬用字元(wildcards)可以帶來十分便利的操作。(不過若使用不當，也是惡夢的開始)

如果各位以前有過 DOS 的操作經驗，應該還記得 \* 及 ? 所代表的意義吧?!

以下是 bash 中使用的萬用字元：

?	代表任何單一字元(character)
*	代表任何字串（注意：0 個以上的字元，例：*yes 將包含 yes 或 yes-or-not）
[字元組合]	在中括號中的字元皆符合，如：[a-z]代表所有的小寫字母
[!字元組合]	不在中括號中的字元皆符合，如：[!0-9]代表非數字的字元皆符合

## 七. 輸入和輸出與重新導向.

當 Linux 系統完成開機之後，預設上，便開有三個檔案，這三個檔案是做為輸入、輸出以及顯示錯誤之用的.

我們稱之為：

- 標準輸入：通常是鍵盤，檔案代碼為 0
- 標準輸出：通常是螢幕，檔案代碼為 1
- 標準錯誤：通常標準輸出相同(也就是螢幕)，檔案代碼為 2

雖然系統已幫你設好了這三個檔案，但我們仍然可以視需要，適時地改變輸入，輸出，及錯誤這三者至不同的地方。這種改變標準輸出入的動作，我們稱之為“**I/O 重新導向**” (I/O Redirection).

例如：

`ls -la > myfile` 就是將查詢的結果重新導向至 `myfile` 中(本來是應該出現在螢幕上的)

`cat myfile` 便可以看見 `ls -la` 的結果.

`cat < myfile > youfile` 就是將 `myfile` 的內容拷貝給 `youfile`.

> 代表將輸出轉向

< 代表將輸入轉向

另一個會將輸出入轉向的機制是“**管線**” (Pipelines).

所謂的管線就是將一個程式的輸出當成另一個程式的輸入.

例如: `cat /etc/passwd | grep ols3`

上面這段指令的意思是說: 把 `/etc/passwd` 檔的內容顯示結果(即輸出) 丟給 `grep` 這個指令當作輸入值, 然後由 `grep` 從中找出包含關鍵字 `ols3` 的資料列.

## 八. 前景與背景工作.

Linux 是多人多工的作業系統, 這意謂 Linux 可以讓多人同時使用, 更可以同時執行許多程式.

一般而言, 你所執行的指令會一直握著控制權, 一直到程式結束為止, 我們稱為這樣的執行工作是在前景工作(foreground jobs), 如果, 執行指令時, 你仍然可以再做其它的事情, 那我們就稱它是在背景工作.

通常比較耗時間的工作, 我們會把它丟到背景去執行, 而這期間, 我們仍然可以和 shell 繼續溝通, 下達其它命令給 shell 去執行.

例如: 我們想從中研院的 FTP 伺服器下載某一個目錄中所有的檔案, 但又不想等它執行完畢(因為這樣耗時間, 也很無聊), 可以用以下的方式來達成:

```
ncftp -R ftp://linux.sinica.edu.tw/pub1/redhat/powertools &
```

(註: 新版 ncftp 已無 `-R` 選項, 請改用 `ncftpget`)

其中 `&` 這個符號便是將命令列丟到背景去執行的指令.

如果您想離線之後, 仍然令系統繼續傳檔, 可以在前面再加一個 `nohup` 的指令, 如下:

```
nohup ncftp -R ftp://linux.sinica.edu.tw/pub1/redhat/powertools &
```

`nohup` 原意: 是 `no hangup` 即不掛斷之意. 在這裡指:離線不中斷指令

## 八. 特殊字元及引號.

有許多字元, 對 shell 來說, 是具有特殊意義的. 詳列於下:

符號	意義
~	自家目錄

`	命令取代
#	註解
\$	變數取值
&	背景工作
*	萬用字元
(	子 shell 開始
)	子 shell 結束
\	使特殊字元恢復本意
	管線
[	字元組合開始
]	字元組合結束
{	命令區塊開始
}	命令區塊結束
;	命令分隔號
'	單引號(不具變數置換的功能)
"	雙引號(具置換的功能)
<	輸入轉向
>	輸出轉向
/	路徑分隔號
?	萬用字元
!	管線邏輯意義上的 NOT

## 九. 常用控制組合鍵.

我們在操作 Linux 時，常會使用一些組合鍵來控制 shell 的活動.

詳列如下：

組合鍵	意義
Ctrl - C	中止目前的命令
Ctrl - \	同上
Ctrl - D	輸入結束，即 EOF 之意（如使用 mail 信件結束時）；或 logout 登出 Linux
Ctrl - Z	暫停目前的命令
Ctrl - M	相當按 Enter

Ctrl - S	暫停螢幕輸出
Ctrl - Q	恢復螢幕輸出
Ctrl - U	將命令列整列刪除
Ctrl - ?	刪除最後一個字元，相當於按 Del

## 十. 指令練習.

指令	選項或參數	意義
ls		
pwd		
which		
more		
less		
passwd		
man		
cat		
touch		
cd		
mkdir		
rmdir		
cp		
rm		
head		
tail		
wc		
grep		
ps		

# Cygwin 環境

對許多人而言，Windows 是他們比較習慣的平台，若要他們為了學習 shell 入門再去學習如何安裝一套作業系統，可能是不小的工程。鑑此，推薦 [紅帽公司](#) 的 Cygwin，這是 Windows 平台上的一種 Unix 環境，在此簡便的環境下嘗試 Linux 及 Shell 的滋味，真是方便極了。（對有心教導中小學生認識 Open Source 的教師而言，這也是很有用的一種教學環境，不必重灌主機）

[cygwin.com](#) 這個網站有提供線上安裝服務，往後缺了什麼套件，都可以在線上補裝，十分便利，且套件為數眾多，版本也都不舊喔。

安裝法如下：

1. 連至 [cygwin.com](#)，如下圖所示：

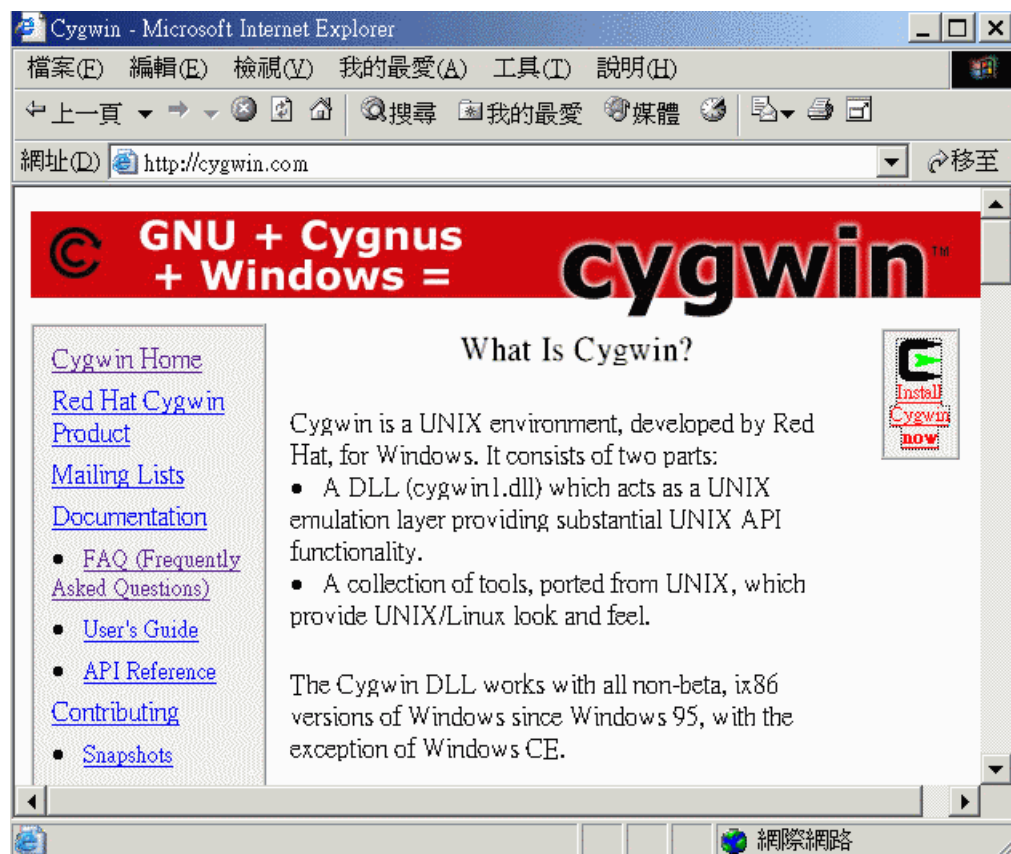


Figure 1. [cygwin.com](#) 首頁

2. 點按畫面右方的 Install Cygwin now，然後選“開啟”以執行線上安裝程式





Figure 2. Install Cygwin now

3. 接著點選安裝套件的 mirror 站台 (交大 <ftp.nctu.edu.tw> 也列名其中喔!), 點按 view 選欲安裝的套件, 如下圖所示:

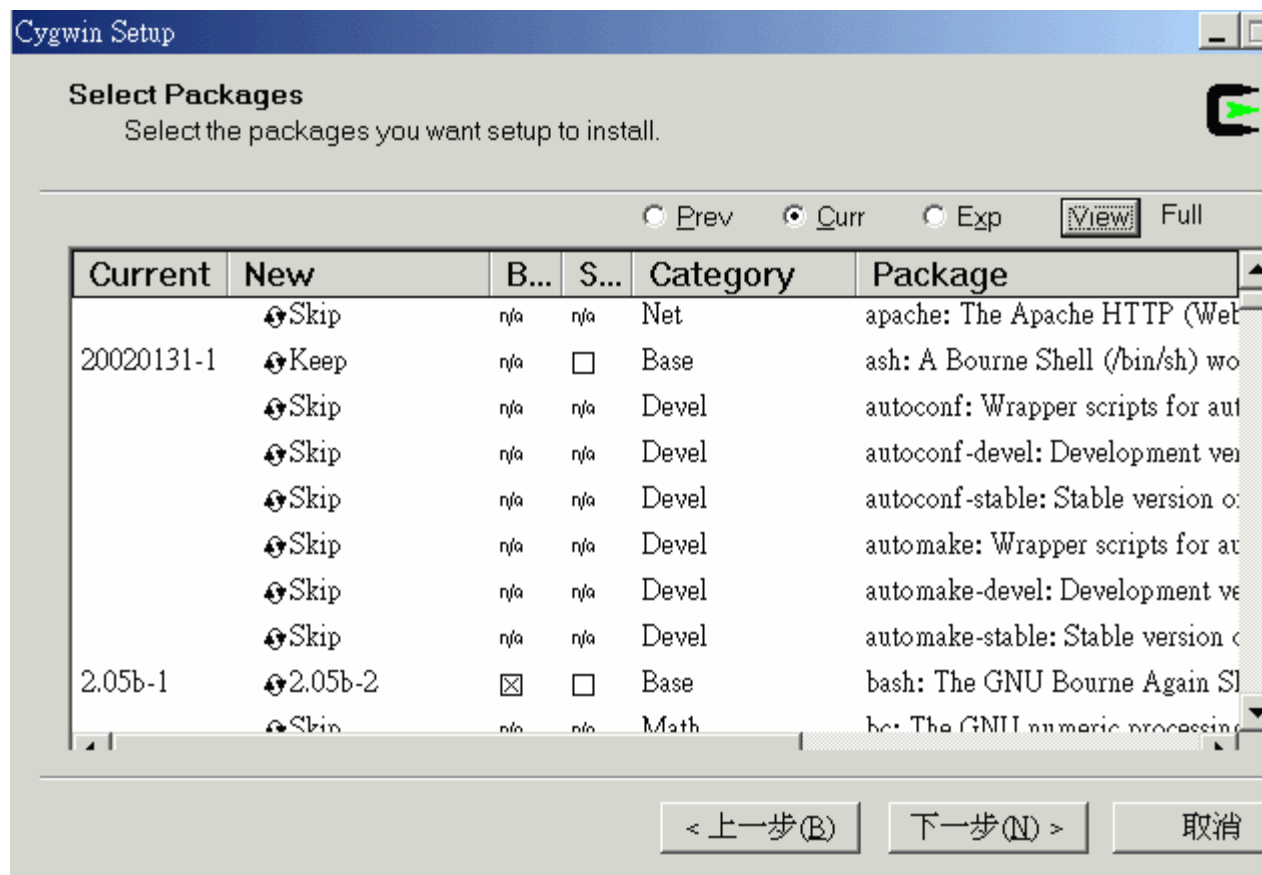


Figure 3. 安裝套件

4. 安裝完成之後, 會在桌面上出現 Cygwin 的 icon 圖示:

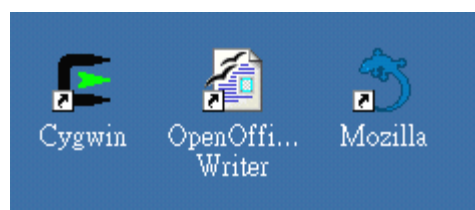
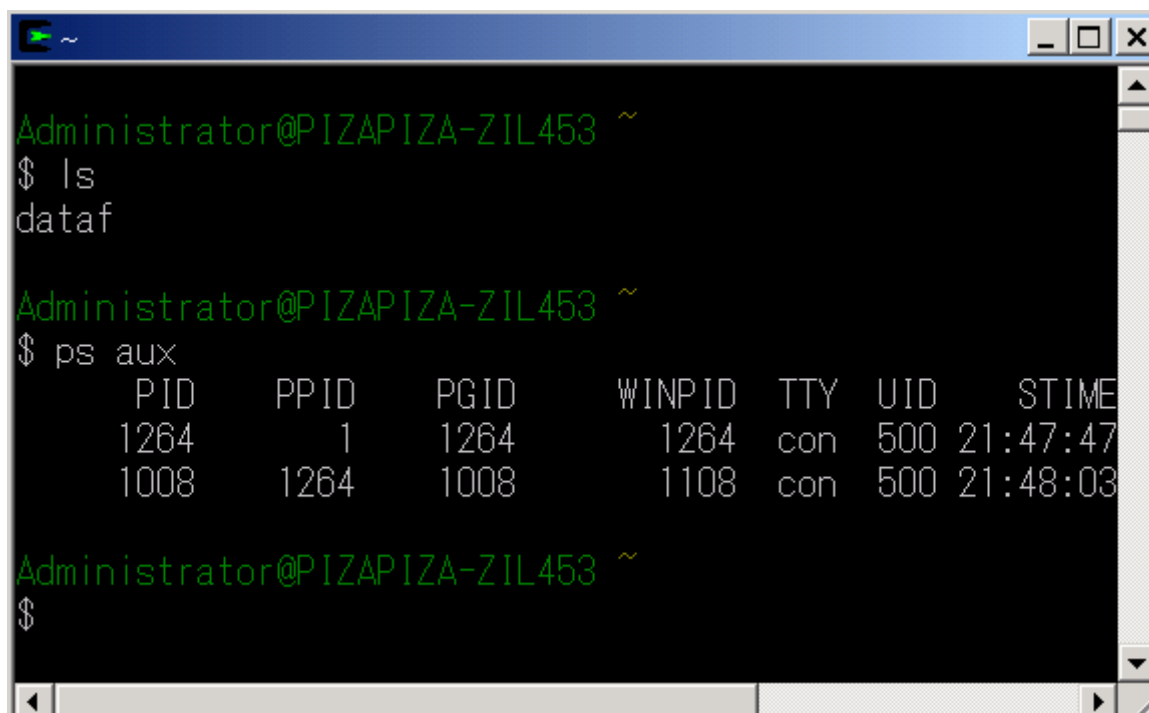


Figure 4. Cygwin 的 icon 圖示

5. 點按之，即可執行此一 Unix 環境。



```
Administrator@PIZAPIZA-ZIL453 ~  
$ ls  
dataf  
  
Administrator@PIZAPIZA-ZIL453 ~  
$ ps aux  
      PID     PPID    PGID      WINPID   TTY   UID    STIME  
      1264         1    1264      1264   con    500  21:47:47  
      1008    1264    1008      1108   con    500  21:48:03  
  
Administrator@PIZAPIZA-ZIL453 ~  
$
```

Figure 5. 執行 Cygwin

## 3. Bash shell 的結構

Bash shell 程式的結構並不複雜，大抵由：內建命令、shell 的語法結構、函式及其它命令列的程式所組成。

shell 程式我們通常稱為 script，以撰寫執行簡便著稱。

## 簡單的示範程式

我們來看一個簡單的例子：hello.sh

```
#!/bin/sh  
#  
# 用途：這是一個簡單 shell 示範程式  
#  
# 第一列以 #! 開頭，表示將用該 /bin/sh 程式來解譯這個 script 檔，  
#
```

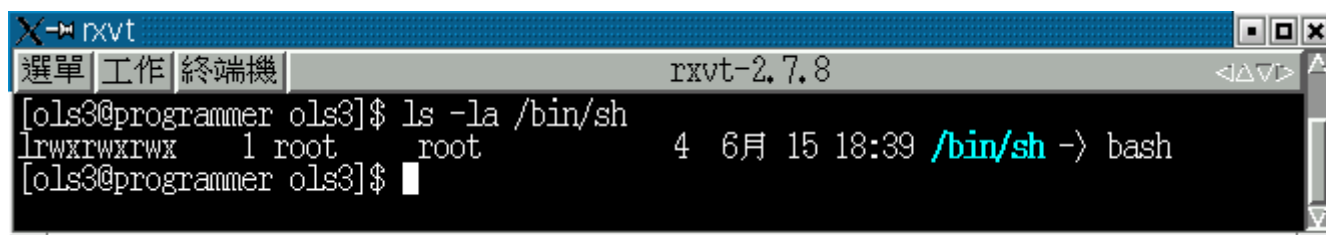
```
# 其它以 # 開頭，則為註解，bash 不予處理。  
#
```

```
echo 'Hello! World!'
```

我們再來看一個簡單的例子：whour.sh

```
#!/bin/sh  
#  
# 用途：這是一個簡單 shell 示範程式  
#  
# 第一列以 #! 開頭，表示將用該 /bin/sh 程式來解譯這個 script 檔，  
#  
# 其它以 # 開頭，則為註解，bash 不予處理。  
#  
# 在 Linux 中 /bin/sh 其實是一 soft link，它連結到 /bin/bash 程式（註  
1）  
#  
  
name="$1"  
ip="163.26.197.1"  
today=`date`  
  
if [ $# != 1 ]; then  
    echo "Usage: ./ $0 [使用者名稱]"  
    exit  
fi  
  
echo "今個兒是 $today，您 $name 大大，來自 $ip"  
  
sleep 5  
  
clear  
  
echo  
echo "Bye-Bye ;-)"
```

註 1:



```
rxvt
選單 工作 終端機 rxvt-2.7.8
[ols3@programmer ols3]$ ls -la /bin/sh
lrwxrwxrwx 1 root root 4 6月 15 18:39 /bin/sh -> bash
[ols3@programmer ols3]$
```

Figure 1. /bin/sh 是 /bin/bash 的 soft link

## 打開執行權

編輯完 shell script 檔之後，通常我們會給它執行的權限：

```
chmod +x hello.sh 或者 chmod 755 hello.sh
```

```
chmod +x whour.sh 或者 chmod 755 whour.sh
```

## 執行 script 的方法

執行的方式有二種：

./hello.sh 或 sh hello.sh 也行，後者倒不必事先給它設定執行的權限。

./whour.sh 或 sh whour.sh 也行，後者倒不必事先給它設定執行的權限。

shell script 撰寫所需條件十分簡便，其實是很容易上手的。

註：script 執行時，現行的 shell（稱為父 shell）會開啟一個子 shell 環境，此 script 即是在這個子 shell 中執行，我們也可以讓 script 在現行的 shell 中執行，方法如下：

```
. hello.sh
```

或

```
source hello.sh
```

另，我們可以追蹤 script 執行的過程，方法如下：

```
sh -x hello.sh
```

這個 -x 即是要進行追蹤之意

## 4. 基本的命令

接著，我們來看一些基本的命令吧。有內建的及命令列程式二種

是不是內建命令，可以用 'type 命令' 看出來，如：

```
type echo 結果為 echo is a shell builtin, 表示它為內建命令
```

```
type mkdir 結果為 mkdir is /bin/mkdir, 表示它為命令列程式
```

## bash shell 的內建命令

bash shell 的內建命令不必再去搜尋路徑中去尋找，直接就可執行，速度很快。

常見的有：

```
alias, bg, bind, break, builtin, case, cd, command, compgen,  
complete, continue, declare, dirs, disown, echo, enable, eval,  
exec, exit,  
export, fc, fg, for, getopts, hash, help, history, if, jobs, kill, let,  
local,  
logout, popd, printf, pushd, pwd, read, readonly, return, set, shift,  
shopt,  
source, suspend, test, times, trap, type, typeset, ulimit, umask,  
unalias,  
unset, until, wait, while
```

以下介紹幾個簡單命令的用法

### echo

用途：echo 用來顯示一系列文字

用法：

1. `echo Hello world`

結果顯示: `Hello world`

它會自動換行；也可以用單引號或雙引號把字串括起來：

```
echo "Hello world"
```

```
echo 'Hello world'
```

2. 若不想它自動換行，可加上 `-n` 的選項

```
echo -n 'Hello world'
```

3. 其它進一步的用法，請 `man echo` 查閱線上文件。

## **cd**

用途：改變目錄位置

用法：

1. `cd`

結果：回到自家目錄(home directory)

效果同於 `cd ~`

2. `cd -`

結果：回到先前的目錄

3. `cd ..`

結果：回到上一層目錄

4. `cd /usr/local/bin`

結果：進入 `/usr/local/bin` 這個目錄

## **pwd**

用途：顯示目前的工作目錄

用法:

1. pwd

結果: 顯示目前所在的目錄位置

```
/home/ols3/shell
```

## alias

用途: 顯示及設定程式別名

用法:

1. alias

結果: 顯示目前所有的已經設定的程式別名

如下所示:

```
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias
--show-dot --show-tilde'
```

2. alias new='old'

結果: 設定程式別名

如下所示:

```
alias cp='cp -f'
```

把 cp 用 cp -f 重新定義, 執行 cp 即等於執行 cp -f

3. unalias 別名

結果: 取消程式別名

如下所示:

```
unalias cp
```

把 cp 的別名設定取消

## 取得變數的內容

\$變數

```
echo $myname
```

```
echo $yourname
```

注意！若變數的周圍尚有其它字元，須用 { } 隔開：

```
echo Hi${myname}KkKk
```

前一章曾提過自動安裝 script，它有個缺點，即版本是固定寫死的，現在我們來加以改良：

```
#!/bin/sh

# 設定版本編號及 MySQL 安裝路徑
ApacheVersion="1.3.26"
PHPVersion="4.1.2"
MYSQLHOME="/home/mysql"

# 解壓
tar xvzf apache_${ApacheVersion}.tar.gz &&
tar xvzf php-${PHPVersion}.tar.gz &&

# 設定 Apache
echo "Configure apache ...." &&
cd apache_${ApacheVersion} &&
./configure --prefix=/usr/local/apache &&

# 設定/編譯/安裝 PHP
cd .. &&
cd php-${PHPVersion} &&
./configure \
    --with-apache=../apache_${ApacheVersion} \
    --with-mysql=$MYSQLHOME &&
make &&
make install &&
cd .. &&
```



```
# 設定/編譯/安裝 Apache
cd apache_$(ApacheVersion) &&
./configure \
    --prefix=/usr/local/apache \
    --activate-module=src/modules/php4/libphp4.a &&
make &&
make install &&

# 拷貝 php.ini 到 /usr/local/lib
cd ../php-$(PHPVersion) &&
cp -f php.ini-dist /usr/local/lib/php.ini

echo
echo "Done!"
echo
```

## 變數的有效範圍

變數一旦設定之後，只有該 shell 的環境中有效，它無法影響其它 shell 環境中的變數，因此，一旦 script 檔執行完畢(該 script 執行時為目前 shell 的子 shell)，該 script 中的變數即不再有效。

testvar.sh

```
#!/bin/sh

testvar="Hello World"

echo $testvar
```

該 script 執行畢，若 echo \$testvar，則將為空。

testvar.sh

```
父 shell 中:
testVAR="Hello World"

testVAR.sh:
#!/bin/sh

echo $testVAR
```

該 script 執行時(變成子 shell)，無法取得父 shell 中的變數值。

```
cd. sh
```

```
#!/bin/sh

cd /usr/local
pwd
```

該 script 執行時，欲切換目錄位置至 /usr/local，唯執行畢，路徑仍不會改變。因為該 script 在子 shell 中執行，當執行完畢時，子 shell 也隨即結束，又回到原父 shell 的環境中，因此路徑不會被改變。

如何達到此一程式構想呢？很簡單，只要讓該 script 在現行 shell 中執行即可：

```
. cd.sh 或 source cd.sh
```

## 使變數成為環境變數

那麼，變數要如何處理，才能為子 shell 取用呢？

當子 shell 產生時，它會繼承父 shell 的環境變數等條件，因此，只要使變數成為環境變數，就能為子 shell 取用。變成環境變數的方法是使用 export 這個命令。

```
testVAR="Hello World"
export testVAR
```

或

```
export testVAR="Hello World"
```

則上述 testVAR.sh 即可取得 testVAR 這個變數的值了。

export 若單獨執行，會列出目前所有的環境變數：

```
export

declare -x COLORFGBG="15;default;0"
declare -x COLORTERM="rxvt-xpm"
declare -x DISPLAY=":0.0"
declare -x GS_LIB="/home/ols3/.kde/share/fonts"
declare -x GTK_RC_FILES="/etc/gtk/gtkrc:/home/ols3/.gtkrc"
declare -x HISTSIZE="1000"
```

```

declare -x HOME="/home/ols3"
declare -x HOSTNAME="programmer.tnc.idv.tw"
declare -x INPUTRC="/etc/inputrc"
declare -x KDE_MULTIHEAD="false"
declare -x KDE_STARTUP_ENV="programmer.tnc.idv.tw;1028124857;691068;872"
declare -x LANG="zh_TW.Big5"
declare -x LC_CTYPE="zh_TW.Big5"
declare -x LESSOPEN="|/usr/bin/lesspipe.sh %s"
declare -x LOGNAME="ols3"
declare -x LS_COLORS=""
declare -x MAIL="/var/spool/mail/ols3"
declare -x OLDPWD="/home/ols3/write/shell-intro"
declare -x
PATH="/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/home/ols3/bin:/usr/java/j2sdk1.4.
declare -x PWD="/home/ols3/write/shell-intro/samples"
declare -x QTDIR="/usr/lib/qt3-gcc2.96"
declare -x SESSION_MANAGER="local/programmer.tnc.idv.tw:/tmp/.ICE-unix/889"
declare -x SHELL="/bin/bash"
declare -x SHLVL="3"
declare -x SSH_ASKPASS="/usr/libexec/openssh/gnome-ssh-askpass"
declare -x TERM="vt102"
declare -x USER="ols3"
declare -x WINDOWID="35651587"
declare -x XDM_MANAGED="/var/run/xdmctl/xdmctl-:0,maysd,mayfn,sched"
declare -x XMODIFIERS="@im=xcin"
declare -x testVAR="Hello world"
~~~~~

```

由上面的列表，可看出 testVAR 已變成環境變數囉。

另外，也不難發現：export testVAR 其實就是 declare -x testVAR

## 由標準輸入讀取資料

（標準輸入通常是鍵盤）可以由鍵盤讀取使用者輸入的資料值：

```

echo "請回答 Y/N ?"
read ans
echo
echo "您的回答是：$ans"
echo

=====

```

```
echo "請輸入二個單字?"  
read X Y  
echo  
echo "您輸入的是: $X $Y"  
echo
```

註：read 不只可以讀取字元，其實字串也可以，換言之，  
上式若使用者輸入 msg yes，則 \$X 為 msg、\$Y 為 yes

另，若輸入的項目比位置變數多，則最後一個變數會吃掉多出來的輸入：

```
echo "請輸入三個字串"  
read X Y Z
```

假設使用者輸入 msg yes ok no today  
則 X 為 "msg"、Y 為 "yes"、Z 則吃掉剩下所有的，即 Z 為 "ok no today"

## 7. 標準輸入/輸出/錯誤 及 I/O 轉向 與 管線

系統一開始預設開啟三個檔案：標準輸入(檔案代碼為 0)，通常連結到鍵盤；標準輸出(檔案代碼為 1)，通常為螢幕；標準錯誤(檔案代碼為 2)，通常也是螢幕。

不過，這些預設的連結是可以改變的，這就是 I/O 轉向的觀念。

比如：echo "Hello world" 原是顯示在螢幕上的，但 echo "Hello world" > test，就可將 "Hello world" 這字串轉向存到 test 中。此 > 即表示輸出轉向，它會開啟一個新檔，若該檔原就存在，則會先被清空。

若不想 test 被清空，可使用 echo "Hello world" >> test，它可以將 "Hello world" 這字串轉向附加到 test 中。此 >> 即表示輸出轉向附加，它會把轉向的內容附加在該檔之後。

cat < test 即是輸入轉向，原應是由鍵盤輸入的，現在則由 test 這個檔案來提供。

假設檔名 z 開頭的檔案不存在，則 ls z\* 將會出現錯誤訊息：

```
ls: z*: 沒有此一目錄或檔案
```

我們可以將此錯誤訊息轉向，而存放在 `error.msg`：`ls z* 2> error.msg`

另外，我們經常會將標準錯誤伴隨標準輸出予以轉向，使執行過程不會產生任何訊息：

如：`/usr/sbin/ntpdate stdtime.sinica.edu.tw 2>&1 > /dev/null`

上述指令的意思是說：把執行過程可能產生的錯誤訊息伴隨著標準輸出(執行後的訊息)，都轉向到垃圾桶 `/dev/null` 中，使之消失不見！

註：在轉向符號 `>` 之後若接的不是檔名，而是檔案代碼(如標準輸入的檔案代碼為 `1`)，則需使用 `>&` 的寫法

在 `Un*x` 系統中，經常會利用幾個小工具，互相搭配，以完成一件工作，這種觀念稱為管線(Pipeline)

比如：`date | cut -d' ' -f3` 就是把 `date` 的輸出，變成 `cut` 的輸入，藉由二者的合作，完成一件工作。這在 `shell script` 中經常可見。

## 本節習題

1. `test` 為一文字檔，試問以下二者有何不同？

```
cat < test
```

```
cat test
```

## 10. 算術運算

Bash shell 的算術運算有四種方式。

1. 第一種：使用 `expr` 這個外部程式

```
加法 r=`expr 4 + 5`
```

```
echo $r
```

注意！`'4'` `'+'` `'5'` 這三者之間要有空白

注意！以下是錯誤的寫法：

```
r=`expr 4 * 5`
```

原因：`*` 對 `bash` 而言有特殊意義(萬用字元)，所以要改用以下寫法：

乘法 `r=`expr 4 \* 5``

也就是說用 `\` 取消 `*` 的特殊意義。

## 2. 第二種：使用 `$(( ))`

上述計算，可用以下方式來做：

```
r=$(( 4 + 5 ))
```

```
echo $r
```

## 3. 第三種：使用 `$[ ]`

上述計算，可用以下方式來做：

```
r=$(( 4 + 5 ))
```

```
echo $r
```

## 4. 乘法

```
r=`expr 4 \* 5`
```

```
r=$(( 4 * 5 ))
```

```
r=$(( 4 * 5 ))
```

```
echo $r
```

## 5. 除法

```
r=`expr 40 / 5`
```

```
r=$(( 40 / 5 ))
```

```
r=$(( 40 / 5 ))
```

```
echo $r
```

## 6. 減法

```
r=`expr 40 - 5`
```

```
r=$(( 40 - 5 ))
```

```
r=$(( 40 - 5 ))
```

```
echo $r
```

#### 7. 求餘數

```
r=$(( 100 % 43 ))
```

```
echo $r
```

#### 8. 乘冪 (如 2 的 3 次方)

```
r=$(( 2 ** 3 ))
```

```
r=$(( 2 ** 3 ))
```

```
echo $r
```

註: expr 沒有乘冪

Bash 的第四種算術運算方法，它使用 `let` 這個命令，如下所示：

加法：

```
n=10
let n=n+1
echo $n
```

結果為 11

乘法：

```
let m=n*10
echo $m
```

結果為 110

除法：

```
let r=m/10
echo $r
```

求餘數：

```
let r=m%7  
echo $r
```

乘冪：

```
let r=m**2  
echo $r
```

雖然 Bash shell 有數種算術運算方法，但並不是每一種方法都可以跨平台，若您的 script 檔要在其它平台上使用，建議最好用 `expr` 這種方式，或許可攜性會好一點。

另外，我們在 script 中經常需要把某一變數做加一運算，以下四法皆可：

```
m=$(( m + 1 ))
```

```
m=`expr $m + 1`
```

```
m=$(( $m + 1 ))
```

```
let m=m+1
```

## 11. 參數傳遞

Bash shell 提供位置參數來擔任參數的傳遞工作。

例： `./pa.sh A B C D E F G H I`

其中 A B C D E F G H I 為欲傳入 pa.sh 這支 script 的 9 個參數，此時，用 `$0` 來代表 script 本身的檔案名稱，`$1` 代表第一個參數，也就是 A，`$2` 代表第二個參數，也就是 B，其它依此類推。

`$#` 代表參數的總數，也就是 9。

`$*` 代表所有的參數，也就是 A B C D E F G H I

`$@` 代表所有的參數，也就是 A B C D E F G H I



`$*` 和 `$@` 稍有不同：`"$*" 代表 "A B C D E F G H I"`，`"$@" 代表 "A" "B" "C" "D" "E" "F" "G" "H" "I"`

`shift` 用來移動參數的位置，每執行一次，則所有的參數往左移一位，此時 `$1` 為 `B`，`$2` 為 `C`，.....

`set` 用來重新設定位置參數，比如：`set a b c d`，表示重新設定 `$1` 為 `a`，`$2` 為 `b`，`$3` 為 `c`，`$4` 為 `d`

pa.sh

```
#!/bin/sh
echo "\$1 = $1"
echo "\$2 = $2"
echo "\$3 = $3"
echo "\$4 = $4"
echo "\$5 = $5"
echo "\$6 = $6"
echo "\$7 = $7"
echo "\$8 = $8"
echo "\$9 = $9"
```

要特別注意的是：位置參數只有 `$1 ~ $9` 等九個，沒有 `$10`。`$10` 是代表 `$1` 和 `0` 的組合，也就是說：若 `$1` 為 `A`，則 `$10` 為 `A0` 之意，例如以下 script 所示：

pa2.sh

```
#!/bin/sh
echo "\$10 = $10"

./pa2.sh A
結果：
$10 = A0
```

第五章曾提過自動安裝 script，現在我們再加以改良：

```
#!/bin/sh
# 使用法： ./install-app.sh Apache 版本 PHP 版本 MySQL 安裝路徑

if [ $# -ne 0 -a $# -ne 3 ]; then
    echo "使用法： ./install-app.sh Apache 版本 PHP 版本 MySQL 安裝路徑"
    echo "或"
```

```
    echo "./install-app.sh"
    exit
fi

# 設定版本編號及 MySQL 安裝路徑

if [ -n $1 ]; then
    ApacheVersion="$1"
else
    ApacheVersion="1.3.26"
fi

if [ -n $2 ]; then
    PHPVersion="$2"
else
    PHPVersion="4.1.2"
fi

if [ -n $3 ]; then
    MYSQLHOME="$3"
else
    MYSQLHOME="/home/mysql"
fi

# 解壓
tar xvzf apache_${ApacheVersion}.tar.gz &&
tar xvzf php-${PHPVersion}.tar.gz &&

# 設定 Apache
echo "Configure apache ...." &&
cd apache_${ApacheVersion} &&
./configure --prefix=/usr/local/apache &&

# 設定/編譯/安裝 PHP
cd .. &&
cd php-${PHPVersion} &&
./configure \
    --with-apache=../apache_${ApacheVersion} \
    --with-mysql=$MYSQLHOME &&
make &&
make install &&
cd .. &&

# 設定/編譯/安裝 Apache
```

```

cd apache_$(ApacheVersion) &&
./configure \
    --prefix=/usr/local/apache \
    --activate-module=src/modules/php4/libphp4.a &&
make &&
make install &&

# 拷貝 php.ini 到 /usr/local/lib
cd ../php-$(PHPVersion) &&
cp -f php.ini-dist /usr/local/lib/php.ini

echo
echo "Done!"
echo

```

## 12. 示 A 兵ノ北 if 糸猿挡篤

Bash shell cm尤ウ示 A 糸 ē 妓 T 轟く臻示北 if 糸猿

T 贺糴猿

```

if ln 庵
then
    ln 1
    ln 2
    .....
fi
if ln 庵; then
    ln 1
    ln 2
    .....
fi

丁
if [ 3 -gt 2 ]; then
    echo ' 3 > 2 '
fi

```

暑琤弧 璜 if cm then 璜 ㄗ ㄱ 珉璜ノ 筵柞

琤 狡慢 璉 if 糸猿

```
if ln 癒; then
```

```
ln 1
```

```
ln 2
```

```
.....
```

```
else
```

```
ln a
```

```
ln b
```

```
.....
```

```
fi
```

```
⌋
```

```
if [ 3 -gt 12 ]; then
```

```
echo ' 3 > 2 '
```

```
else
```

```
echo ' 3 < 12 '
```

```
fi
```

```
⌋: 86.sh
```

```
#!/bin/sh
```

```
# 糺ノ 磅へセ祢 A 琿 ノタ紘 よ A?
```

```
# ノ猿: ./86.sh 計 卩 ㊦
```

```
# -ne 琿い単ゝ種
```

```
if [ $# -ne 1 ]; then
```

```
echo "ノ猿 $0 計 卩 ㊦"
```

```
exit
```

```
fi
```

```
# 璜才 A 玊 𠃊 块 挡獮
```

```
echo "Great!"
```

```
echo "昵块 琿 $1"
```

```
if ln 癒
```

```
then
```

```
ln 1
```

```
ln 2
```

```
.....
```

```
elif ln 癒
```

```
then
```

```
ln 1
```

```
ln 2
```

```
.....
```

```

elif ln    掩
then
    ln    1
    ln    2
    .....
else
    ln    a
    ln    b
    .....
fi

```

└

```

if ln    掩; then
    ln    1
    ln    2
    .....
elif ln    掩; then
    ln    1
    ln    2
    .....
elif ln    掩; then
    ln    1
    ln    2
    .....
else
    ln    a
    ln    b
    .....
fi

```

└

```

if [ "$a" -gt "$b" ]; then
    echo " $a > $b "
elif [ "$a" -eq "$b" ]; then
    echo " $a = $b "
else
    echo " $a < $b "
fi

```

└: 87. sh

#! /bin/sh

```

#  厶ノ  磅厶セ标 A  厶ノタ紘  よ A?
#  厶ノ猿: ./87.sh  计  1  计  2
#  -ne  厶い单厶え種

if [ $# -ne 2 ]; then
    echo "厶ノ猿  $0  计  1  计  2"
    exit
fi

#  瓚  条猿,  玊祚...  厶!

if [ "$1" -gt "$2" ]; then
    echo "$1 > $2"
elif [ "$1" -eq "$2" ]; then
    echo "$1 = $2"
else
    echo "$1 < $2"
fi

echo "Done!"

=====

计  厶耕  厶龄才腹

-gt  厶

-eq  单厶

-ne  い单厶

-lt  厶

-ge  厶厶单厶

-le  厶厶单厶

```

## 13. 真假值判断

何者为真？何者为假？ Bash shell 以其傳回值來決定，若傳回 0 則為真，傳回非 0 為假。

命令的傳回值會儲存一個特殊的變數裡：\$?

例：[ 3 -gt 2 ]

echo \$? 會傳回 0，因為 3 的確大於 2

例：[ 3 -gt 12 ]

echo \$? 會傳回 1，因為 3 不大於 12

在此 [ ] 代表真假值判斷，[ 和 3 、 12 和 ] 之間必需要有空白；-gt 是 greater then 之意，即大於。

真假值的判斷有三大類：檔案測試、字串比較以及整數比較，以下列示之：

```
-a file
    若檔案存在則為真

-b file
    若檔案存在且是一個特殊的 block 檔則為真

-c file
    若檔案存在且是一個特殊的 character 檔則為真

-d file
    若檔案存在且是一個目錄則為真

-e file
    若檔案存在則為真

-f file
    若檔案存在且是一個正規的檔案則為真

-g file
    若檔案存在且其 set-group-id 位元已設定則為真

-h file
    若檔案存在且是一個符號連結檔則為真

-k file
    若檔案存在且其 "sticky" 位元已設定則為真

-p file
    若檔案存在且是一個 named pipe (FIFO) 則為真
```

`-r file`  
若檔案存在且具有讀取的權限則為真

`-s file`  
若檔案存在且檔案大小大於 0 則為真

`-t fd`  
若檔案代碼 fd 已開啟且連接到一個終端機則為真

`-u file`  
若檔案存在且 s set-user-id 位元已設定則為真

`-w file`  
若檔案存在且具有寫入的權限則為真

`-x file`  
若檔案存在且具有執行的權限則為真

`-0 file`  
若檔案存在且被有效的使用者 id 所擁有則為真

`-G file`  
若檔案存在且被有效的群組 id 所擁有則為真

`-L file`  
若檔案存在且是一個符號連結檔則為真

`-S file`  
若檔案存在且是一個 socket 則為真

`-N file`  
若檔案存在且自它上次被讀取之後已被修改過了則為真

`file1 -nt file2`  
若 file1 比 file2 新則為真

`file1 -ot file2`  
若 file1 比 file2 舊則為真

`file1 -ef file2`  
若 file1 和 file2 有相同的設備和 inode 編號則為真

`-o optname`  
若 shell 的選項名稱 optname 已設定打開則為真



`-z string`

若字串長度為 0 則為真

`-n string`

若字串長度不是 0 則為真

`string1 == string2`

`string1 = string2`

若字串 `string1` 和 `string2` 相同則為真

`string1 != string2`

若字串 `string1` 和 `string2` 不相同則為真

`string1 < string2`

若字串 `string1` 排序比 `string2` 小則為真

注意！字串小於比較的正確寫法為 [ `"$str1" \< "$str2"` ]

因為 `<` 對 Bash 而言是輸入轉向之意，所以要放一個跳脫字元 `\` 使其特殊意義消失！

`string1 > string2`

若字串 `string1` 排序比 `string2` 大則為真

注意！字串大於比較的正確寫法為 [ `"$str1" \> "$str2"` ]

因為 `>` 對 Bash 而言是輸出轉向之意，所以要放一個跳脫字元 `\` 使其特殊意義消失！

`arg1 -eq arg2`

若整數 `arg1` 等於 `arg2` 則為真

`arg1 -ne arg2`

若整數 `arg1` 不等於 `arg2` 則為真

`arg1 -lt arg2`

若整數 `arg1` 小於 `arg2` 則為真

`arg1 -le arg2`

若整數 `arg1` 小於或等於 `arg2` 則為真

`arg1 -gt arg2`

若整數 `arg1` 大於 `arg2` 則為真

`arg1 -ge arg2`

若整數 arg1 大於或等於 arg2 則為真

若 要用 < > <= >= 來比較數字，則判斷式需擺在 (( )) 中，如：

```
(( "982" > "24" ))
```

藉由邏輯運算，可結合好幾個真假值判斷：

```
[ -r filename1 -a -x filename ]
```

若 filename1 可讀且可執行則為真，-a 即 '且' 之意。

```
[ -r filename1 -o -x filename ]
```

若 filename1 可讀 或 可執行 則為真，-o 即 '或' 之意。

```
[ ! -r filename1 ]
```

若 filename1 不是可讀 則為真，! 即 '非' 之意。

常用的判斷技巧：

```
if [ "$var" = "" ]; then
    echo "$var 是空字串"
fi
```

以下效果相同：

```
if [ ! "$var" ]; then
    echo "$var 是空字串"
fi
```

```
if [ -z "$var" ]; then
    echo "$var 是空字串"
fi
```

```
if [ "X${var}" = "X" ]; then
    echo "$var 是空字串"
fi
```

## 14. case 彙彖挡篤

璚璫 糶 贺薄徧 い ノ case 彙彖

```
case 糶 in
    v1) ln 1
```

```

        ln 2
        ln 3
        ;;
v2) ln 1
    ln 2
    ln 3
    ;;
v3) ln 1
    ln 2
    ln 3
    ;;
v4) ln 1
    ln 2
    ln 3
    ;;
*) ln 1
   ln 2
   ln 3
esac

```

┐

```

echo -n "叫块 昵匡兜(1/2/3/4) "
# パ夹非块 弄      ㄗ  opt  い
read opt

case "$opt" in
    1) echo "1 ㄗ      !"
        ;;
    2) echo "2 ㄗ M    !"
        ;;
    3) echo "3 ㄗ睺    !"
        ;;
    4) echo "4 ㄗ      !"
        ;;
    *) echo "$opt ㄗ 匡兜 1~4 條瞅"
esac

```

## 15. 迴圈 語法結構

迴圈通常用在重覆執行，直到某一條件成立或失敗為止。

有三種迴圈：for、while、until，說明如下：

## for 的迴圈

for 迴圈格式：

```
for 變數 in 範圍
do
    命令 1
    命令 2
    .....
done
```

例：

```
for char in A B C D E F G
do
    echo "$char"
done
```

例：以下在 /home/us1 ~ /home/us4 的目錄下，開啟一個 index.html 檔

```
for usr in /home/us[1-4]; do

    # 開啟 html 目錄做為網頁存放空間
    htmldir="$usr/html"

    # 若目錄不存在，則開啟一個新目錄
    if [ ! -d "$htmldir" ]; then
        mkdir -p "$htmldir"
    fi

    # 產生 index.html
    echo "您是 $usr" > "$usr/html/index.html"

    # 改變擁有者為該使用者
    chown -R "$usr.$usr" "$htmldir"

done
```

注意：

1. 記得使用 root 權限來做
2. 本例可再改寫成：具自動開啟帳號/網頁空間、幫使用者寫好第一個首頁檔 的

## 功能

另，Bash 也有以下迴圈寫法：

```
for ((i=1; i<=10; i++))
do
    echo "$i"
done
```

注意！`i=1; i<=10; i++` 須放在二個 `()` 之中

```
#
# 大量建帳號的實列：
#

#!/bin/sh

# 適合大量開設有規則的帳號名字

for ((n=1; n<=10; n++))
do
    username="st$n"
    adduser $username
    mkdir -p "/home/$username/html"
    echo "這是您 $username 的第一個首頁檔 index.html" >
"/home/$username/html/index.html"
    chown -R "$username.$username" "/home/$username/html"
    echo "已開設 $username"
done
```

```
# 適合沒有規則的帳號名字
# 假設 teachers.txt 內含教師的帳號，如下所示：
ya1
mo6
po2
yu8
ru0
```

以下程式可以大量來處理沒有規則的帳號

```

#!/bin/sh
r=`cat teachers.txt`

for th in $r
# 上式就等於 for th in ya1 mo6 po2 yu8 ru0 一樣
do
    username="$th"
    adduser $username
    mkdir -p "/home/$username/html"
    echo "您 $username 的第一個首頁檔是 index.html" >
"/home/$username/html/index.html"
    chown -R "$username.$username" "/home/$username/html"
    echo "已開設 $username"
done

#
# for 的無窮迴圈：
#

for ((;;))
do
    echo "Hi"
    sleep 2
done

或

# 只要讓條件判斷永遠為真，即可形成無窮迴圈。

for ((i=1; i>0; i++))
do
    echo "Hi"
    sleep 2
done

```

## 16. 函式

我們可以把常用的程式片斷寫成一個區塊，這個區塊具有完成特定動作的功效，並且賦予它一個簡單的名稱為代表，往後要使用這個特定功效時，只要呼叫此一區塊名稱即可，像這種機制，我們稱之為函式。

# 函式的寫法

```
function 函式名稱 () {  
    命令 1  
    命令 2  
    .....  
}
```

例：

```
function Hello () {  
    echo "Hello World"  
}
```

Hello

則可呼此一函式，並秀出 Hello World

注意！function 或 () 可以省略其中一個，但不可以二個都省略。

以下二種寫法都可以：

```
function 函式名稱 {  
    命令 1  
    命令 2  
    .....  
}
```

```
函式名稱 () {  
    命令 1  
    命令 2  
    .....  
}
```

函式也可以指定傳回值，方法如下：

```
函式名稱 () {  
    命令 1  
    命令 2  
    .....  
    return n  
}
```

其中 return 即是要傳回值之意，n 為 0 ~ 255

## 17. select 匡虫条猿

select よ猿妝 房 匡虫

## select 耀猿

```
#!/bin/sh

select fn in *; do
    # 瑣い 匡兜い, 玥瞞
    if [ -z $fn ]; then
        break
    fi
    echo "$fn (you choose $REPLY)"
done

# 瑣 select の done い丁△T break, 玥穢い 枷 loop
# 匡兜计 硃 $REPLY い

=====

挡獬 :

1) dataf          5) Factor2.sh    9) function.sh  13) sysinfo.sh
2) exp.sh         6) Factor3.sh    10) picts       14) test2.sh
3) Factor0.sh     7) Factor.sh     11) select.sh
4) factor2.sh     8) f.sh          12) sel.sh
#? 2
exp.sh (you choose 2)
1) dataf          5) Factor2.sh    9) function.sh  13) sysinfo.sh
2) exp.sh         6) Factor3.sh    10) picts       14) test2.sh
3) Factor0.sh     7) Factor.sh     11) select.sh
4) factor2.sh     8) f.sh          12) sel.sh
#?
```

## 18. 歴史記録



Bash shell 有支援歷史記錄功能，它會把您過去鍵入的命令，記錄在某一檔案中(`~/.bash_history`)，供您往後再次取用。

歷史記錄的筆數則視環境設定的大小而定(HISTSIZE 這個環境變數定義其大小)，預設值是 1000 筆。history 這個指令，可以秀出這 1000 筆記錄，每筆記錄都有編號，按佇列的方式排列，新的記錄會將舊記錄往前擠出。換言之，您看到的記錄可能是 18 ~ 1017 (共 1000 筆)，而不一定每次都是由 1 ~ 1000

## 取用歷史記錄的方法

取用歷史記錄的方法如下：

先用 `history | more` 來觀看舊指令的記錄，如下所示：

```
1301  which uuencode
1302  rpm -qf /usr/bin/uuencode
1303  which uuencode
1304  ls
1305  ls -la dataf5
1306  cat dataf5
1307  uuencode dataf5 dataf5 > e5.txt
1308  vi e5.txt
1309  clear
1310  uuencode dataf5 HELLO > e5.txt
1311  vi e5.txt
1312  clear
1313  cd ..
1314  cd ..
1315  cd work
1316  ls
1317  clear
1318  ls -la *.gz | more
1319  uuencode cjkmix2.tar.gz cjkmix2.tar.gz > m.txt
1320  vi m.txt
1321  clear
1322  uuencode cjkmix2.tar.gz cjkmix2.tar.gz | mail -s "give u a file"
ols3@localhost
1323  mail
1324  vi m2.txt
1325  vi m2.txt
1326  clear
1327  uudecode m2.txt -o cc.tar.gz
1328  ls -la c*.gz
```

1329 clear

擇定之後(比如：您要的是第 1312 clear 這道指令)，只要下 !1312 即可再次執行 clear

! 1322 則可再次執行 uuencode cjkmix2.tar.gz cjkmix2.tar.gz | mail -s "give u a file" ols3@localhost

## 實例

1. 將大寫檔名改成小寫 或 將小寫檔名改成大寫

```
#!/bin/sh
#
# 將大寫檔名改成小寫檔名 或 將小寫檔名改成大寫檔名
# 至於使用那一個功能，完全由執行時的檔名來決定
#
# 這支程式本身不會被改名
#
# 安裝法：
# 1. cp low2upper /usr/local/bin
# 2. cd /usr/local/bin
# 3. ln -sf low2upper upper2low
#
# Copyright 2002 OLS3(ols3@www.tnc.edu.tw)

#
# Functions
#
# low2upper : 將小寫檔名改成大寫檔名
# upper2low : 將大寫檔名改成小寫檔名
#

low2upper () {
    for FILE in *
    do
        if [ "$FILE" != "$LFNAME" -a "$FILE" != "$UFNAME" ]; then
            mv -i "$FILE" `echo "$FILE" | tr ' [a-z]' '[A-Z]'` 2> /dev/null
        fi
    done
}
```

```

upper2low () {
    for FILE in *
    do
        if [ "$FILE" != "$LFNAME" -a "$FILE" != "$UFNAME" ]; then
            mv -i "$FILE" `echo "$FILE" | tr ' [A-Z]' '[a-z]` 2> /dev/null
        fi
    done
}

#
-----

#
# main
#

LFNAME=low2upper
UFNAME=upper2low
NOWFILENAME=`basename $0`

if [ "$NOWFILENAME" = "$LFNAME" ]; then
    low2upper
elif [ "$NOWFILENAME" = "$UFNAME" ]; then
    upper2low
fi

echo
echo "Done!"
echo

```

2. 找出網路卡的 IP (請用 root 權限來執行)

```

#!/bin/sh

# @(#)fip.sh
#
# $Id$
# $Author$
#
# Copyright (C) 2002 by OLS3 All rights reserved
#

# 取得 IP 最終版

```

```

interface="$1"

if [ -z "$interface" ]; then
    echo "Usage: $0 [eth0/eth1...]"
else
    if dmesg | grep -q "$interface"; then
        ifconfig "$interface" | sed -n 's/inet addr:./p' | awk '{ print
$1 }'
    else
        echo "$interface not found!"
    fi
fi

# 為何 if dmesg | grep -q "$interface"; then 不寫成

# if [ dmesg | grep -q "$interface" ]; then

# 因為 if dmesg | grep -q "$interface" 就會傳回 0 或非 0 之值，不必再
用 [ ] 來測試

```

### 3. 5-1/5-2 習題參考解答： 簡易樂彩程式

```

#!/bin/sh

# 取得一個 1 ~ 42 的號碼
function GetLOTO () {
    r=$(( $RANDOM % 43 ))

    # 若該號為 0，則予以加 1
    if [ $r -eq 0 ]; then
        r=$(( r + 1 ))
    fi

    # 使每一個號碼都用二位數來表示，不足 10 者，在其前面補 0
    if [ $r -le 9 ]; then
        echo "0$r"
    else
        echo $r
    fi
}

```

```

function GenNumAndCheckRepeat () {
    # 取得 6 個號碼，並且予以排序
    m=`{ GetLOT0; GetLOT0; GetLOT0; GetLOT0; GetLOT0;
GetLOT0; } | sort -n`

    # 把這 6 個號碼暫時 copy 一份給 n
    n="$m"

    # 檢查這 6 個號碼有無重覆
    n=`echo $n | tr ' ' '\n' | uniq -d`

    # 若 $n 為空，表示沒有重覆，則傳回真 ( 0 )
    if [ -z "$n" ]; then
        return 0
    else
        # 若 $n 非空，表示有重覆，則傳回假 ( 1 )
        return 1
    fi
}

if [ $# -ne 1 ]; then
    echo "使用法: $0 組數"
    exit
fi

if [ "$1" -lt 1 -o "$1" -gt 99 ]; then
    echo "使用法: $0 [1-99]組"
    exit
fi

i=1
while [ $i -le "$1" ]
do
    # 取得 6 個號碼
    GenNumAndCheckRepeat

    # 若有重覆，則丟棄這 6 個號碼
    if [ $? -ne 0 ]; then
        continue
    fi

    # 調整輸出格式

```

```
j=$i
if [ $j -le 9 ]; then
    echo -n "第 0$j 組: "
else
    echo -n "第 $j 組: "
fi

# 執行到此，表示該組號碼沒有重覆，予以顯示出來
echo $m

# 組數 加 1
i=$(( i + 1 ))
done
```

## cygwin note

*本網頁以打造無障礙閱讀為目標，可以用任何瀏覽器來觀看本網頁*

---

- [簡介](#)
- [下載與安裝](#)
- [網路資源](#)

### 簡介

簡單的說，**cygwin** 在 **windows** 上提供了一個像 **linux (Linux-like)** 的環境，可以讓使用者在 **windows** 上執行 **linux** 的程式。

嚴格地說，**cygwin** 是模擬了 **GNU** 的環境，而不是 **linux** 的環境，藉著 **cygwin** 的函式庫(library)，在 **Win32 API** 環境上提供了像 **linux** 般的 **API** 環境。利用 **Cygwin**，你可以：

- 將 **linux** 上的程式移植到 **Windows**，而不必做重大的修改，只需將原始碼拿來作 **configure** 與 **make** 就好了。
- 使用常用的 **linux** 工具程式，如 **grep**, **sed**, **awk** 等。
- 撰寫 **Win32 native console** 或是 **GUI** 應用程式。

## 發展歷史

**Cygwin** 發展於 1995 年, **Cygnus Solutions** 公司 (已為 **redhat** 所併購) 以自由軟體基金會的 **gnu** 工具為基礎，將它移植到 **windows** 上，它一直都是自由軟體。初期只有命令列視窗 **bash** 及編譯器 **gcc** 等等開發工具可以使用；現在則連 **X** 視窗系統都已移植成功，目前正將 **X** 視窗系統下的應用軟體逐漸移植過來。

## 下載與安裝

1. 連至 [www.cygwin.com](http://www.cygwin.com) 下載 [setup.exe](#)
2. 執行 **setup.exe**，選取 **Download from Internet**
  1. 選取距離自己最近的下載點
  2. 選取所需套件 // 建議全選 **devel**, **lib** 等分類的所有套件
3. 執行 **setup.exe**，選取 **Install from Local directory** 安裝套件 // 最好選取 **All user**
  1. 選取所有套件
4. 執行桌面上的 **Cygwin** 的 **icon** 圖示即可執行此一 **Cygwin** 環境。// 視窗大小可由 **icon** 圖示右點按 選取調整字型 及佈置

step by step 可參考

<http://linux.tnc.edu.tw/techdoc/shell/x52.html>

## 設定

### 顯示中文

安裝好 **cygwin** 之後，發現無法在 **console** 看到中文，必須做以下設定

#### 1. ~/.bashrc

```
stty cs8 -istrip
stty pass8
export LANG=C
export LC_CTYPE=iso-8859-1
alias ls="ls --show-control-chars"
alias crxvt='rxvt -fn "細明體-16" -fn "細明體-16" -km big5 -e bash
--login -i'
```

#### 2. ~/.inputrc

```
set meta-flag on
set input-meta on
set convert-meta off
set output-meta on
```

### 參考:

- <http://sources.redhat.com/ml/cygwin/2001-07/msg00311.html>
- <http://www2.cs.uh.edu/~kcting/blog/index.cgi/opencontent/cygwin/CygwinHOWTO.html>



## VI 中的 Backspace 與 Delete 按鍵異常

首先確定 \$TERM 的設定為 cygwin

```
echo $TERM
```

若不是，請設定為

```
TERM = cygwin  
export TERM
```

Backspace 與 Delete 這兩個鍵正常狀況下只有一個有用。可在 \$HOME/.bash\_profile 任選一個來使用

```
stty erase '^H' //ctrl+v, ctrl+h  
  
stty erase '^?'
```

參考：

- <http://www.mgt.ncu.edu.tw/~dino/unix/editor03.htm>
- <http://www.ibb.net/~anne/keyboard/keyboard.html#Bash>
- <http://ece.niu.edu.tw/~chu/download/doc/UNIX.txt>

## ls 無法看見顏色

可編輯 /etc/bashrc 或家目錄下的 .bashrc ，加入下列指令

```
alias ls='ls -F -N --color=auto'
```

要立即生效的話，則執行下列命令

```
source /etc/bashrc
```

或

```
source .bashrc
```

## 應用

### 在 Cygwin 上安裝 ssh 伺服器

1. OS修改環境變數：path增加d:\cygwin\bin
2. OS增加環境變數：CYGWIN=ntsec tty
3. Cygwin安裝必要的套件：OpenSSH, cygrunsrv
4. Cygwin 設定 OpenSSH:
  1. 執行 ssh-host-config
  2. Should privilege separation be used? (yes/no) : yes
  3. Should this script create a local user 'sshd' on this machine? (yes/no) : yes
  4. Do you want to install sshd as service? (yes/no) : yes
  5. Default is "ntsec" . CYGWIN=ntsec tty
5. 手動啟動 OpenSSH 服務：net start sshd
6. 手動關閉 OpenSSH 服務：net stop sshd

### 參考：

<http://pigtail.net/LRP/printsrv/cygwin-sshd.html>

## 網路資源

- [Cygwin: 送自由入微軟帝國](#)
- [Cygwin Hints and Tips](#)