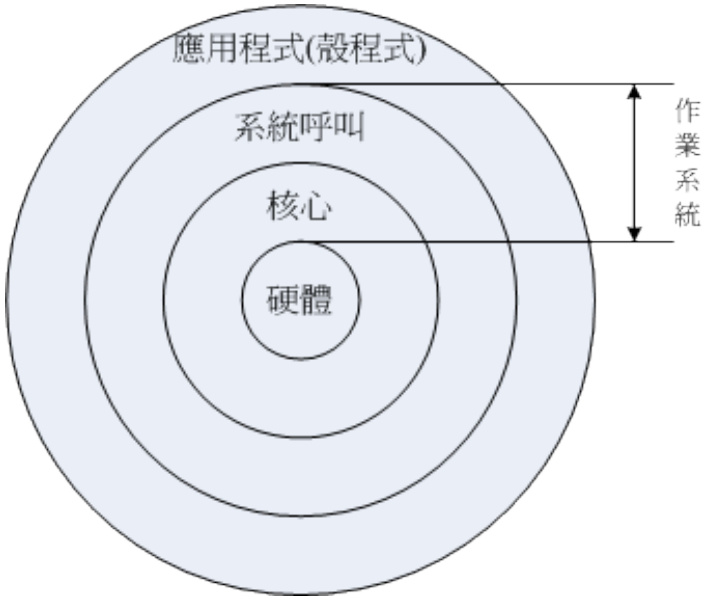
bash shell 基礎認識

Reading material

- bash shell 基礎認識
  - http://linux.vbird.org/linux_basic_train/unit07.php#7.1
- **認識與學習BASH**
  - http://linux.vbird.org/linux_basic/0320bash.php

何謂shell



- 登入系統取得的文字型互動界面就稱為 shell
- 我們可以切換不同的shell
- 系統內可用的shell都在 /etc/shells 這個檔案
  - /bin/sh (已經被 /bin/bash 所取代)
  - /bin/bash (就是 Linux 預設的 shell)
  - /bin/tcsh (整合 C Shell ，提供更多的功能)
  - /bin/csh (已經被 /bin/tcsh 所取代)
- 預設用(登入後)的shell
  - /etc/passwd 裡面，使用冒號 (:) 分隔的第 7 個欄位

例題：

- 使用『 echo $0 』查閱這個變數以及其輸出的內容
- 使用『 /bin/sh 』切換 shell 成為 sh
- 使用『 echo $0 』查閱這個變數以及其輸出的內容
- 使用pstree觀查bash與sh程序的觀係
- 透過『 exit 』離開 sh之後，再次echo $0觀察目前的 shell 名稱為何？

/sbin/nologin shell

- 無法登入的shell

- 許多系統預設要執行的軟體，例如 mail 的郵件分析、WWW 的網頁回應等等, 會建立帳號給這些軟體使用，但這些帳號不需要登入，就可以給這些帳號nologin shell

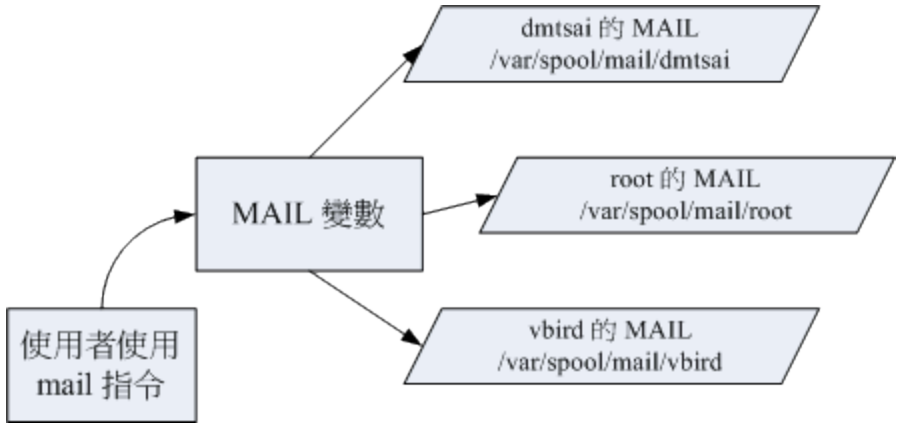
例題：

- 請使用 usermod 來修改 一般帳戶的 shell 變成 /bin/sh
- 檢查/etc/passwd，看一般帳戶的資訊有什麼不同
- 嘗試使用一般帳戶登入，看看會出現什麼情況

- 注意: 不要改到root
- 請使用 usermod 來修改 一般帳戶的 shell 變成 /sbin/nologin
- 嘗試使用一般帳戶登入，看看會出現什麼情況

- 請再次以 usermod 的方式將 shell 改回來 /bin/bash

Shell 變數的功用



無變數的情況下，若要訂正程式，每個地方都要更改

…/var/spool/mail/user
……/var/spool/mail/user…
/var/spool/mail/user…
……
……/var/spool/mail/user……
……

有變數的情況下，最上方的username更改一下，後面的通通變動了

username=/var/spool/mail/user
…$username
……$username…
$username…
……
……$username……
……

Shell 變數

```
[student@localhost ~]$ 變數="變數內容"
[student@localhost ~]$ echo $變數
[student@localhost ~]$ echo ${變數}
```

- 變數與變數內容以一個等號『=』來連結
- 等號兩邊不能直接接空白字元
- 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
- 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
  - 雙引號內的特殊字元如 $ 等，可以保有原本的特性
  - 單引號內的特殊字元則僅為一般字元 (純文字)
- 可用跳脫字元『 \ 』將特殊符號(如 [Enter], $, \, 空白字元, ' 等)變成一般字元
- 在一串指令的執行中，還需要藉由其他額外的指令所提供的資訊時，可以使用反單引號『`指令`』或『$(指令)』。
- 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
- 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
- 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)
- 取消變數的方法為使用 unset ：『unset 變數名稱』

例題：

- 設定一個名為 myname 的變數，變數的內容為『 peter pan 』，使用 echo 呼叫出 myname 的內容

- 設定 hero 變數的內容為『 I am $myname 』，其中 $myname 會依據 myname 變數的內容而變化。設定完畢請呼叫出來。

- 設定 kver 變數，內容為『 my kernel version is 3.xx 』，其中 3.xx 為 uname -r 輸出的資訊。請注意， kver 變數設定過程中，需要用到 uname -r 這個指令的協助。

例題：

- 使用『 find /usr/bin -perm 4755 』找出所有含有特殊權限的檔名
- 使用『 ls -l $(find /usr/bin -perm 4755) 』將所有檔名的權限列出

環境變數

變數	功能
LANG LC_ALL	語系資料，例如使用 date 輸出資訊時，透過 LANG 可以修改輸出的訊息資料。
PATH	執行檔搜尋的路徑～目錄與目錄中間以冒號(:)分隔，由於執行檔/指令的搜尋是依序由 PATH 的變數內的目錄來查詢，所以，目錄的順序也是重要的。
HOME	代表使用者的家目錄，亦即使用者看到的 ~ 代表的目錄。
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)。
HISTSIZE	這個與『歷史命令』有關。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。
RANDOM	『隨機亂數』的變數。目前大多數的 distributions 都會有亂數產生器，亦即 /dev/random 檔案。讀者可以透過這個亂數檔案相關的變數 ($RANDOM) 來隨機取得亂數。在 BASH 的環境下，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。
PS1	命令提示字元，可使用 man bash 搜尋 PS1 關鍵字，即可了解提示字元的設定方式。
?	$? 這個變數內容為指令的回傳值，當回傳值為 0 代表指令正常運作結束，當不為 0 則代表指令有錯誤。

路徑相關的環境變數

- 若該變數為擴增變數內容時，則可用 "$變數名稱" 或 ${變數} 累加內容
  - PATH=/myhome/bin:$PATH

PS1 環境變數

- 命令提示字元的長像
  - \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『study.centos.vbird』
  - \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『study』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『dmtsai』；
  - \v ：BASH 的版本資訊，如鳥哥的測試主機版本為 4.2.46(1)-release，僅取『4.2』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～

$? 與 int main()的關係

在 C 的標準文件裡面有一段話是這樣，

> ***5.1.2.2.1 Program startup***
>
> *The function called at program startup is named main. The implementation declares no prototype for this function. It shall be defined with a return type of int and with no parameters:*

```
int main(void) {
    /* ... */
}
```

> *or with two parameters (referred to here as argc and argv, though any names may be used, as they are local to the function in which they are declared):*

```
int main(int argc, char *argv[]) {
    /* ... */
}
```

> *or equivalent[1]; or in some other implementation-defined manner.*

其中就有提到 `main()` **這隻程式的傳回值需要是** `int` **型態。**
如果不是的話， ~~迂腐~~**標準**的 C compiler 會給你/妳口頭警告（但是還是可以編譯出執行檔），見下圖。

```
#include <stdio.h>
int main()
{
 printf("Hello, World!");
 return 0;
}
```

區域/全域變數、父程序與子程序

- 區域變數
  - 變數只能在目前這個 shell 當中存在，不會被子程序所沿用
- 全域變數
  - 變數會儲存在一個共用的記憶體空間，可以讓子程序繼承使用。
- 兩隻 bash 之間僅有全域變數 (環境變數) 會帶給子程序， 而子程序的變數，是不會回傳給父程序的。
- 若該變數需要在其他子程序執行，則需要以 export宣告變數
- env : 顯示環境變數
- Set: 顯示所有變數

例題：

- 使用 set 或 env 觀察是否存在 mypp 這個變數？
- 設定 mypp 的內容為『 from_ppid 』，用echo顯示出來
- 使用 set 或 env 觀察是否存在 mypp 這個變數？

- 執行『 /bin/bash 』進入下一個 bash 的子程序環境中
- 使用 set 或 env 觀察是否存在 mypp 這個變數？同時說明為什麼？
- 設定 mypp2 的內容為『 from_cpid 』，並且呼叫出來
- 使用『 exit 』離開子程序回到原本的父程序，觀察是否存在 mypp2 這個變數？為什麼？

- 使用『 export mypp 』後，使用 env 或 export 觀察是否存在？
- 執行『 /bin/bash 』進入下一個 bash 的子程序環境中
- 使用 set 或 env 觀察是否存在 mypp 這個變數？同時說明為什麼？