第十一章 進階外殼操作

11-1 外殼與核心

『外殼』(Shell) 是一套程式，它會隨時監督使用者輸入命令。當使用者輸入命令並敲入『Enter 鍵』之後，Shell 立即解譯該命令並執行，完成之後，Shell 再回去繼續等待接受命令。Shell 包含許多『外殼命令』(Shell command)，使用者透過這些 Shell command 可以操作系統，或要求完成某一特定工作。之前，我們大略介紹過 Unix/Linux 是屬於開放性系統。這裡所說的開放系統，是指『系統核心』(System kernel) 是開放的，亦即核心內系統函數 (System call) 的原始碼都是開放的，需要遵循各自使用授權方式，如 GPL。系統函數即是操作該系統最基本的介面程式，譬如，操作檔案系統的系統函數有 open(file_name)、read(file_ID, data)、write(file_ID, data)等等。既然系統函數是開放的，任何人都可利用這些函數編寫應用系統，因此造成 Unix/Linux 系統上的應用軟體可以大量且快速的被發展出來，甚至其中許多功能強大的系統套件都屬於自由軟體，也造成軟體工業更快速的發展 (並非全都免費的，有些還是需要付費購買)。

核心大多與主機硬體或系統操作環境有關程式的結合體，如欲在系統內增加硬體設施，則必須編寫相關驅動程式再嵌入核心內。另外，欲增加系統功能，譬如雙核心 CPU 處理、虛擬記憶體、加密/解密安全、多重執行緒.....等等功能，也必須編寫相關程式並嵌入核心。由此可見，核心程式可能會被隨時呼叫使用，但隨系統功能越強，它就越來越龐大。可能成長到並無法全部儲存於記憶體內，而大部分儲存磁碟機內，需要時再呼叫它並導入記憶體內。雖然核心程式提供有標準介面，但還是需利用程式語言才可以呼叫它，各方領域的專家們，為了方便使用者操作系統，所以利用系統函數發展出許多系統工具。基本上，所謂 Unix/Linux 僅是指系統核心而言，而利用核心的函數所發展出來的工具，因處於系統核心的外部，因此稱之為『外殼』(Shell)。

也就這樣，隨著不同環境的需求，Shell 也出現了幾種不同的版本，如圖 11-1 所示。基本上，任何版本的 Unix/Linux 系統都可以依照需求安裝一個或多個 Shell，甚至可以針對任一使用者的需求，而給予不同的版本。雖然市面上有許多 Shell 版本，但是版本之間大

多能夠相容，在學習方面並不會出現太大的困擾。以下列出幾種較常見的版本：

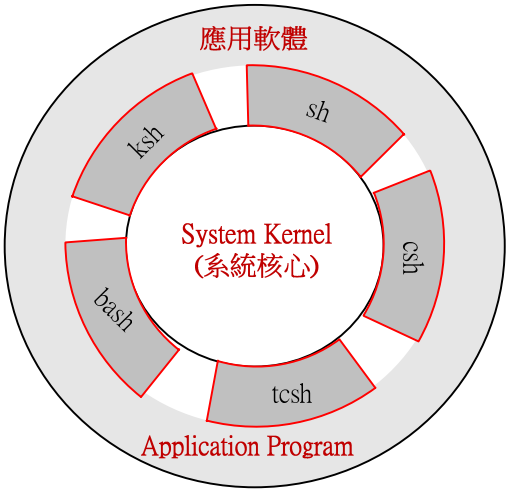

圖 11-1 外殼與系統核心

- **Bourne Shell (sh)：**此為 Unix 系統上最普遍使用的 Shell，是由 AT&T 貝爾實驗室的 Steven Bourne 所編寫出來。自 1979 年後 sh 幾乎是所有 Unix 系統的標準配備。
- **Korn Shell (ksh)：**是由 AT&T 的 David Korn 針對 Bourne Shell 所提出的改良版。第一版的 ksh 在 1986 年出版。
- **Bourne Again Shell (bash)：**此為目前 Linux 系統最普遍使用的版本，是在 1988 年被公開出來，它最初是 Brian Fox 為自由軟體基金會所寫的 GNU 專案，再進一步由 Chet Ramey 開發而成。
- **C Shell (csh)：**是 BSD Unix 系統的標準配備，大多數 Unix 系統都會安裝此套件。C Shell 是由加州大學柏克萊分校的 Bill Joy 所撰寫，於 1979 年被包括在 BSD 系統內。
- **TC Shell (tcsh)：**是 C Shell 的改良版，於 1980 年出版。

基本上，任何一套 Unix/Linux 系統都提供有多種 Shell 執行環境，隨著個人喜好可切換到任何一種 Shell 來執行，都不會影響它的執行結果，因為都是呼叫相同的核心程式。

11-2 Shell 環境運作

每一外殼程式包含著一串列的 Shell 命令，並會建立一個與使用者交談的環境，稱之為『外殼環境』(Shell Environment)。當系統管理者幫使用者建立帳戶時，會幫使用者指定一

個 Shell 環境（如，/bin/sh）。使用者登入系統之後，就會啟動所指定的 Shell，使用者與系統之間便在所指定的 Shell 環境下運作。一般 Shell 環境的運作步驟如下：(如圖 11-2 所示)

1. 出現 提示符號（"$" 記號）準備接受命令。
2. 使用者（或終端機傳輸）輸入命令。
3. 依照 PATH 變數，尋找命令位置。
4. 產生子行程，解譯並執行命令。
5. 輸出結果或錯誤訊息給使用者。
6. 再出現 Prompt 準備接受下一個命令。

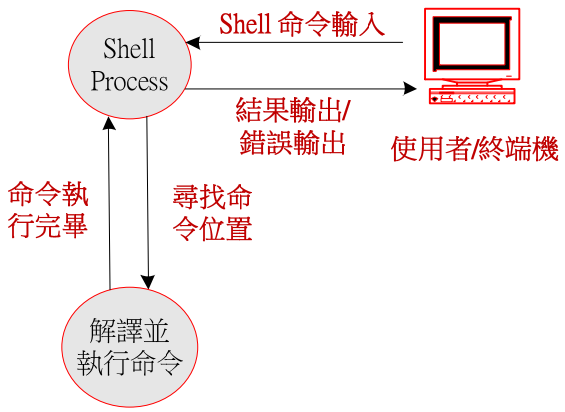

圖 11-2 Shell 環境的運作

當 Shell 執行命令時，Prompt 會消失，一直到該命令執行完畢，才會再出現 Prompt。如果 Prompt 一直不出現，極有可能該命令進入無窮迴圈永遠執行不完。每當 Shell 接受收一個命令時，會依照 PATH 變數搜尋命令位置，範例如下：

```
$ set | grep  PATH
PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/tsnien:.
```

PATH 變數內記錄著多個搜尋命令的目錄位置，每一個欄位紀錄一個搜尋路徑，欄位之間是利用冒號（:）分隔。以上述範例為例，當使用者下達命令之後，系統搜尋 /usr/Kerberos/bin 目錄下是否有該命令的程式，如沒有會再到 /usr/local/bin 目錄下尋找，再找不到則會繼續到/usr/X11R6/bin 的目錄下尋找，依此類推。如果 PATH 所指定的所有目錄都找不到的話，則會出現 "Command not found" 訊息，範例如下：

```
$ ly
-bash: ly: command not found
```

由此可見，出現 “Command not found” 訊息並非所下的命令一定不存在，有可能僅是所指定的 PATH 路徑的問題而已。

11-3 Shell 命令格式

11-3-1 命令格式

每個 Shell 命令可結合許多選項，延伸出更多不同的執行結果，這也是 Shell 最強的地方。當使用者下達某一命令時，可能包含有下列三個內容：

- **命令名稱（command）：** Shell 程式的名稱。
- **選項（options）：** 可依照使用者需求給予不同的選項，執行上述命令時，可得到不同的結果。
- **引數（arguments）：** 執行命令時給予對應的參數。

我們利用最常使用的 ls 命令，來觀察所給予的選項會出現哪些不同的結果，範例如下：

```
$ ls -a -l -r -F   /home/tsnien
total 136
-rw-rw-r--       1 tsnien tsnien    209 Mar 30 10:02 t2.c
-rwxrwxr-x       1 tsnien tsnien 4929 Mar 30 10:02 t2*
-rw-rw-r--       1 tsnien tsnien      70 Mar 31 19:15 t1
……
-rwxrwxr-x       1 tsnien tsnien 4738 Mar 30 09:48 a.out*
-rw-r--r--       1 tsnien tsnien    742 Apr 10    2004 .zshrc
drwxr-xr-x       2 tsnien tsnien 4096 Feb 18 12:10 .xemacs/
-rw-------       1 tsnien tsnien 3100 Apr    7 19:53 .viminfo
drwx------       2 tsnien tsnien 4096 Apr    6 11:11 .ssh/
```

其中所選項有 -a、-l、-r、以及 -F，引數為 /home/tsnien，各選項功能如下：

- -a：列出所有檔案及目錄(含隱藏檔)。
- -l：以『長』列表方式顯示檔案及目錄。
- -r：以相反順序表列。

- -F：增加檔案表示。在每一目錄之後加斜線（/），可執行檔之後加星號（*）。

另外，多個選項也可以組合方式下達，如下：

```
$ ls -alrF    /home/tsnien
```

11-3-2 萬用字元

執行命令當中，經常需要輸入某些引數，引數當中也許有一些共通或相似的地方，可用『萬用字元』（Wildcards）來表示。一般系統大多會使用到下列三種萬用字元：

- **『*』（星號）：**表示任何長度的任何字元。
- **『?』（問號）：**表示一個任何字元。
- **『[]』（中括號）：**表示中括號內其中任何一個字元。

範例說明如下：

命令型態	說　　明
$ ls –l hash_2	顯示 hash_2 檔案的詳細資料。
$ls –l h*	顯示有 h 開頭的任何檔案，如 hash_2、hash_1、hae 等等。
$ls –l *ash*	顯示名稱中有 ash 的任何檔案，如 hash_1 等等。
$ls –l hash_?	顯示 hash_ 開頭且緊接任一字元的檔案，如 hash_1、hash_2 或 hash_3 等等。
$ls –l hash_[123]	顯示 hash_1、hash_2 或 hash_3 檔案。

操作範例如下：

```
$ ls hash_2.pl          【指定檔案名稱】
hash_2.pl
$ ls h*                 【檔案開頭為 h 的任何檔案】
hash_1.pl   hash_2.pl   hash_3.pl   hash_4.pl
$ ls *ash*              【檔案中有 ash 字元的任何檔案】
hash_1.pl   hash_2.pl   hash_3.pl   hash_4.pl
$ ls hash_?.pl          【檔案名稱有 hash_ 後面緊接著任何單一字元】
hash_1.pl   hash_2.pl   hash_3.pl   hash_4.pl
```

```
$ ls hash_[12].pl      【檔案名稱為 hash_1 或 hash_2】
hash_1.pl  hash_2.pl
```

11-3-3 特殊字元使用

一般 Unix/Linux 系統的檔案（或目錄）名稱，大多不允許下列特殊符號出現：

& * \ | [ ] { } $ < > ( ) # ? ‘ “ / : ^ ! ~ % ;

但 Shell 操作方面會使用到某些特殊字元，來增強 Shell 命令的功能，較常用到的特殊字元有：

字　元	範　例	說　明
\	\?	取消後面字元的特殊意義。
‘	‘string’	取消在 string 字串中任何字元的特殊意義。
“	“string”	取消任何字元的特殊意義，除了 $、’ 與 \。
`	`string`	執行在 string 中的指令。

上述特殊字元中，除了反斜線（\）是以單一字元出現外，其他字元都是成雙成對的將某一字串包起來；以下分別說明之。

【\ 反斜線】

有一些特殊字元都有其特殊的功能，譬如，問號（?）表示任何一個字元；星號（*）表示任何長度的任何字元。然而，有時候也希望這些字元能表現出它原來的意義，譬如，問號就是 ?，星號就是 *。在此情況下，就必須利用 \ 字元來取消它特殊字元的意義。範例如下：

```
$ echo * star *
a.out args b_dir dead.letter ex1.c ext2.c f2 Fedora #Fedora# Fedora~ #file-1#
f#
$ echo \* star \*
* star *
```

在第一個 echo 命令，系統將 * 視為任何長度的任何字元；然而在第二個 echo 命令，利用 \ 字元取消 * 的特殊意義。

【’ 單引號】

利用兩個單引號包起來的字串，所有特殊字元的意義都被取消。範例如下：

```
$ echo '*** star ***'
*** star ***
```

上述可以看出，兩個單引號包起來的星號（*），不再代表其他意思。

【` 反向單引號】

反向單引號(`，鍵盤左上角與 “~” 同一按鍵)是 Unix/Linux 系統上常見的特殊符號。兩個反向單引號表示執行內部的指令，範例如下：

```
$ echo Today is =  `date`
Today is = Sun Jun 12 11:23:37 CST 2005
```

上述範例可以看出，反向單引號內的命令 date，被執行後再顯示其執行結果。

【” 雙引號】

一般系統或程式語言較常見到的是兩個雙引號（”）包起來的字串；在 Unix/Linux 系統還是保存著雙引號內表示字串的意思，但也保存某些特殊字元的意義，譬如，$、’ 與 \ 字元。範例如下：

```
$ echo "Today is = `date` "
Today is = Sun Jun 12 11:31:30 CST 2005
```

所以雙引號內的反向單引號還是保留原來的功能。

11-4 Shell 特殊操作

為了提高 Shell 的處理能力，一般 Shell 環境都會提供一些特殊操作的工具，而且在不同 Shell 環境之間，這些特殊工具大致上還是可以相容。

11-4-1 標準輸入/輸出轉向

當執行某一命令時，也許需要輸入某些資料，執行後可能會有結果輸出，或執行當中發生錯誤則需要輸出錯誤訊息告知使用者，Unix 系統都將這些輸入/輸出設定成標準化，如下（如同 C 語言）:

- 標準輸入（stdin，代號 0）: 鍵盤輸入。
- 標準輸出（stdout，代號 1）: 終端機輸出。
- 標準錯誤輸出（stderr，代號 2）: 終端機輸出。

如同 C 語言一般，可以透過設定檔將上述標準輸入/輸出轉移到其他硬體裝置上。有時候為了方便操作，也可直接將標準輸入/輸出轉向到其他檔案；亦即將鍵盤輸入或終端機輸出由另一個檔案來取代它的輸入與輸出。轉向方法如下：

- 『<』：輸入轉向。將鍵盤輸入由另一個輸入（如檔案）取代。
- 『>』：輸出轉向。將輸出到終端機的資料（標準輸出或錯誤輸出）轉向到另一個位置（如檔案）上。
- 『 >>』：輸出附加轉向。將所輸出的資料附加（append）到另一個檔案之後。

以下用幾個範例來說明轉向的功能：

(A) 【輸出轉向 – “>>”】

『輸出轉向』是將原來終端機輸出轉向到另一個裝置（或檔案）上。如果所轉向的目的檔案不存在時，系統會自動建立新檔案並寫入資料；如該檔案存在的話，會寫入並覆蓋原資料（原資料將遺失掉）。範例操作如下：

```
$ cat file_4                          【觀察 file_4 內容】
Good Luck To You !!
$ cat file_4 >file_6                  【將 file_4 連結並轉向輸出到 file_6】
$ cat file_6                          【查閱 file_6 內容】
Good Luck To You !!
$
```

(B) 【輸出附加轉向 – “>>”】

如果轉向目標是已存在的檔案，又希望保存原來資料的話，就必須利用『輸出附加轉向』，它會將新的資料寫在原資料的後面，而且不會覆蓋舊資料。操作範例如下：

```
$ cat file_5                          【觀察 file_5 內容】
How Are You ?
$ cat file_5 >>file_6                 【將 file_5 連結轉向附加在 file_6 後面】
$ cat file_6                          【查閱 file_6 內容】
Good Luck To You !!
How Are You ?
$
```

(C) 【輸入轉向 – “<”】

『輸入轉向』是將原來由鍵盤輸入的資料，轉向由另一個週邊裝置取代，最簡單的輸入轉向是檔案輸入來取代鍵盤輸入，操作範例如下：

```
$ cat   <file_6   >file_7   【將 file_6 連結顯示，並轉向輸出到 file_7】
$ cat   file_7              【查閱 file_7 內容】
Good Lucky To You !!
How Are You ?
$
```

(D) 『終端機複製』

Unix 使用者時常利用 cat 命令將終端機上的顯示複製到某一檔案上，用來產生一個新檔案。這對於產生小檔案非常方便，範例如下：

```
$ cat   >file_4                    【將輸出連結轉向到 file_4】
Good Lucky To You !!
?
[2]+   Stopped                     cat >file_4
```

11-4-2 管道輸入與輸出

『管道』(pipe，|) 是用來連結兩個以上的命令，將前面命令執行的結果，轉向成後面命令的輸入 (command | command)。譬如，$cat file_1 | sort，表示前面命令 cat file_1 的執行結果，經過管道轉向成後面 sort 命令的輸入，至於 sort 命令的輸出結果就送給標準輸出，由終端機顯示出來。上述命令是由大到小排序顯示 file_1 的內容。管道的操作範例如下：

```
$ cat file_2 |sort
adm:x:3:4:adm:/var/adm:/sbin/nologin
amanda:x:33:6:Amanda user:/var/lib/amanda:/bin/bash
apache:x:48:48:Apache:/var/www:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
canna:x:39:39:Canna Service User:/var/lib/canna:/sbin/nologin
csu001:x:501:501::/home/csu001:/bin/bash
csu003:x:504:504::/home/csu003:/bin/bash
csu004:x:505:505::/home/csu004:/bin/bash
....
```

11-4-3 聚集命令輸入

所謂『命令行』(Command line) 操作，表示每一行輸入一個命令，並且以交談方式繼續輸入下一個命令。在許多情況若需要同時輸入多個命令時，則可利用『分號』(;) 將命令聚集在一起。範例如下：

```
$ echo "多命令聚集"                          【單一 echo 命令】
多命令聚集
$ date                                      【單一 date 命令】
Sun Jun 12 14:19:00 CST 2005
$ ls                                        【單一 ls 命令】
level    object    Welcome
$
$ echo "多命令聚集";date ;ls                 【聚集上述三個命令】
多命令聚集
Sun Jun 12 14:18:35 CST 2005
level    object    Welcome
```

上述範例中，首先個別輸入 echo、date 與 ls 三個命令，接著在同一命令行中輸入同樣命令，並利用『分號』(;) 分隔這些命令，發現會得到相同的結果。聚集多個命令也可以利用『轉向』(>)，將輸入轉向到另一個檔案，但必須利用『小括號』(( )) 將這些命令包起來，範例如下：

```
$ (echo "多命令聚集";date ;ls) > file_1
$ cat file_1
多命令聚集
Sun Jun 12 14:28:10 CST 2005
file_1
level
object
Wellcome
```

11-4-4 程序中斷

當使用者下達一個命令之後，Shell 立即產生一個程序 (Process) 來執行該命令，且程序會在執行完該命令隨之消失。然而若一個程序執行很久無法完成時，可利用『Ctrl + c』(先按住 Ctrl 鍵再按 C 鍵) 強迫中斷該程序。我們利用一個 sleep (沉睡) 程序的產生與中斷作為範例來說明，操作如下：

```
$ sleep 30
≧                    【提示符號不見，按入 Ctrl + c 後中斷程序再出現 $】
$
```

沉睡命令 sleep n 表示 Shell 執行一些無意義的程式 n 秒。在執行當中將不會出現提示符

號（$），其中 n 秒也不會很準確。執行中可輸入 Ctrl + c 中斷該命令，其他命令一樣可以利用相同的方法來中斷。

11-4-5 背景執行

基本上，使用者與 Shell 之間是交談式（Interactive）的溝通，使用者下達命令之後，Shell 立即接受命令並執行它，執行當中 Shell 因無暇掃描鍵盤，所以沒有任何輸出到終端機上；此時使用者將無法繼續輸入命令，好像與系統斷了線失去聯絡一般，直到 Shell 執行完命令之後，才會繼續掃描鍵盤，並將執行結果顯示到終端機上，如此所產生的程序稱之為『前景執行』（Foreground process）。如果期望命令執行當中，Shell 能繼續掃描鍵盤，並讓使用者繼續輸入命令的話（Shell 也可以繼續接收並處理命令），則必須將它設定成『背景程序』（Background process），其設定方法只要在命令的後面加入一個『&』記號即可。基本上，背景程序與前景程序的運作情況完全相同，執行完畢之後，都會由標準輸出顯示其結果。接下來利用 jobs（顯示執行工作）、ps（顯示程序狀態）、vi（文書編輯）與 kill（刪除程序）命令，來測試背景程序的運作，操作如下：

- 產生一個背景程序：（$ vi &）

```
$ vi test1.c &
[1] 541
$ vi test2.c &
[2] 542
[1]+  Stopped                 vim test1.c
```

- 觀察目前使用者所下的工作：

```
$ jobs
[1]-  Stopped                 vim test1.c
[2]+  Stopped                 vim test2.c
$ ps –ef | grep tsnien| grep vim
tsnien     541 32726  0 15:57 pts/6    00:00:00 vim test1.c
tsnien     542 32726  0 15:57 pts/6    00:00:00 vim test2.c
tsnien     545 32726  0 15:58 pts/6    00:00:00 grep vim
```

使用者可鍵入 jobs 命令，觀察到自己所下達的背景程式（或工作）。另一種方式是利用 ps（process status）命令觀察系統有哪些執行中的程序，再利用檢蒐命令，搜尋使用者自己所產生的那些程序中，是否有 vim 程序。接下來，利用 kill 命令來刪除所產生的程序，操作範例如下：

```
$ kill -9 541
$ kill -9 542
$ ps -ef |grep tsnien|grep vim
tsnien       576 32726    0 16:05 pts/6       00:00:00 grep vim
$
```

上述範例中，kill 命令是刪除執行中的程序，其中 -9 是最高等級，表示程序無論在何種狀態下，都會被無條件刪除。使用者僅允許刪除自己所產生的程序，刪除後可利用 ps 命令觀察是否程序還存在。

(A) 【背景程序輸出】

背景程序執行完畢之後，如果有輸出的話，會由標準輸出顯示出來 (如沒有另外指定轉向輸出的話)，操作範例如下：

```
$ (sleep 4; echo Good Lucky to You)&
[1] 577
$ Good Lucky to You
                                        【Enter 輸入】
[1]+  Done                          ( sleep 4; echo Good Lucky to You )
$
```

11-5 過濾器

11-5-1 過濾命令彙集

Unix/Linux 系統最有價值的是，外殼程式是自由發展出來的，許多特殊工具也在此情況下被大量發展出來。接下來，我們來探討一些特殊文件處理的工具，它們常被統稱為『過濾器』(Filter)。過濾器從標準輸入 (或轉向輸入) 取得一串資料，以某種方式轉換 (或處理)，並將結果送到標準輸出 (或轉向輸出)，其運作方式如圖 11-3 所示。表 11-1 為較常使用的過濾器工具，接下來幾節 (第 11-7 ~ 11-9 節) 將介紹一些較常用且較複雜過濾器的操作方式；本節先介紹一些過濾器共同操作方法。

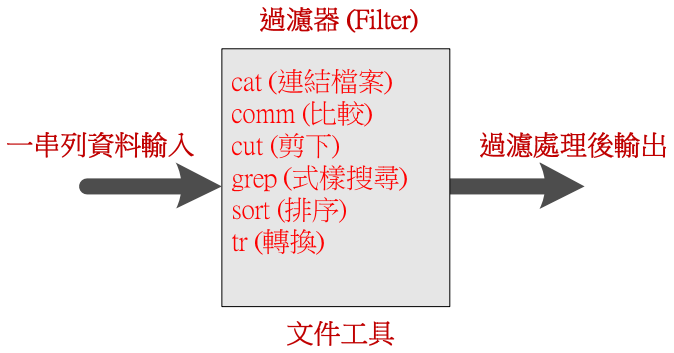

圖 11-3 過濾器功能

表 11-1 Unix/Linux 常用過濾器命令

過濾器	功　　能
cat	(concatenate)。連結並顯示文字。
comm	(compare)。比較排序過的檔案。
crypt	(encryption)。將資料加密編碼或解密。
cut	從輸入列剪下某些欄位。
diff	(differences)。兩檔案之間以行對行比較並顯示不同處。
egrep	使用完整的正規表示式搜尋式樣文字。
fgrep	使用字元字串搜尋文字。
grep	使用式樣搜尋文字。
head	顯示檔案的開頭部份。
less/more	顯示文字，一次一個畫面。
nl	將每一行編號。
paste	將文字行合併。
pg	顯示文字，一次一個畫面。
pr	格式化列印文字。
sort	排序文字。
spell	檢查拼字錯誤。
tail	顯示檔案中最後的部分。
tr	(translate)。轉換文字。
uniq	顯示檔案中的唯一行，移除重複內容的行。
wc	(word count) 計算檔案中的行數、字元數。

11-5-2 聚集過濾器

單一命令的過濾器功能並沒有辦法顯現出來有何特殊，Unix/Linux 過濾器可貴的地方，就在於利用管道（pipeline，|）將許多過濾器命令結合在一起，其處理能力不容小覷。譬如，$ cat /etc/passwd | sort | more，表示 /etc/passwd 檔案內容經由 sort 命令過濾後，再傳送給 more，操作範例如下：

```
$ cat /etc/passwd | sort | more
adm:x:3:4:adm:/var/adm:/sbin/nologin
amanda:x:33:6:Amanda user:/var/lib/amanda:/bin/bash
apache:x:48:48:Apache:/var/www:/sbin/nologin
....
csu020:x:518:518::/home/csu020:/bin/bash
csu021:x:519:519::/home/csu021:/bin/bash
--More--
```

11-5-3 正規表示式

簡單的說，過濾器即是一個檔案（或一連串資料）經過某一特殊功能處理之後，得到所需要的訊息。比如說，檢蒐文件內是否拼字錯誤、替換文件內某一語句、選取某一特殊字元資料、或剪接某片段文字訊息等等。也許讀者會認為這些文件處理工具，應該與系統操作沒有任何關係，而與一般文書處理事務較有關才對；但為何這些過濾器會被號稱為 Unix/Linux 系統最傲人的工具呢？讀者可以試著想看看，如果我們想要了解目前系統狀況，也許必須輸入某些命令來觀察，譬如，輸入 ps 命令觀察系統目前有哪些程序在執行、輸入 nstat 命令觀察目前有哪些網路傳輸埠口被開啟、輸入 who 命令觀察目前有哪些使用者登入系統、甚至必須隨時隨地去閱讀『日誌檔案』（Log file），是否有使用者越權的存取狀態等等。當執行這些命令時，大多會出現一連串的大筆資料，我們再仔細的閱讀這些資料，從中找出所期望的訊息，如此一來，會出現兩個問題：

- 人工檢蒐文件容易疏忽遺漏。
- 必須隨時隨地下達系統管理命令，並檢視其執行結果。

過濾器除了方便操作系統之外，解決上述兩大問題才是它的主要目的。我們可以依照某些特殊需要，編寫 Shell 程式（第七章介紹），其中包含過濾器，並可設定成定期執行程式（cron，第十二章介紹），如此便可以解決上述兩大問題（第十八章介紹）。

既然要由搜尋文件中，焠鍊出所需要的資訊，那應該如何表示所要搜尋的文字（或稱字串，Pattern）？這是一件很大的學問。搜尋文件不可能指定某一字元（或字串），譬如，僅在文件中尋找『RedHat』，而將其替換成『Fedora』，這種過濾功能太為簡單了（但也有許多地方僅這樣用），也沒什麼萃取功能。一般所欲尋找的字串大多是有一些共通性，甚至模糊相似的地方。我們利用這些模糊相似的字串，作為搜尋的依據，如此才能找出較精煉的資訊。而 Unix/Linux 利用工業領域裡常用的『正規表示式』（Regular expression）來表示資料之間共通模糊相似的地方。表 11-2 為較常用的正規表示式，Unix/Linux 系統的各種搜尋、編輯工具都會使用到這些表示法，如 grep、cut、paste、sed、awk 等等文字處理工具。在爾後幾個章節裡，讀者將會看到這些表示式的使用範例。簡略說明如下：

- ^Text：一列開始是 Text 字串。
- Text$：一列的結尾是 Text 字串。
- [xyz]：x、y、或 z 字元皆符合。
- x1|y2：x1 或 y2 字串皆符合。
- x*：一個以上或沒有 x 字元皆符合。
- x+：一個以上的 x 字元皆符合。
- x?：x 再任何一個字元皆符合，如 x1、xa、xt 等等皆符合。

表 11-2 常用的正規表示式

符號	意義	範例	符合字串
.	任何一個字元	chi.	chil, chid
*	任何空白或一個以上的字元	te*ch	tech, teach
[]	符合任何在中括號內出現的字元	[Tt]ea	Tea, tea
[a-z]	符合任何範圍內的字元	[A-Za-z]	任何字元
^	一列的開始	^The	一列開始是 The 字串
$	一列的結尾	book$	結尾是 book 字串

11-6 檔案搜尋工具

檔案搜尋工具就是在檔案（或某一串列資料）內搜尋所要的文字（或字串），這些目標文字可以直接以純文字表示，也可以利用正規式表示。基本上，搜尋工具在一般文書處理器裡常見到，譬如，vi、word、記事本等等。但 Shell 與文書處理兩者的搜尋工具有很大的不同，文書處理大多是將游標放置到目標字串的位置，而 Shell 是處理一連串進來的文字（無論串列資料或檔案皆是如此），找出目標字串，其中並沒有游標工具。一般找到目標字串後，通常會將該行內容顯示於終端機上，或許還會有其他的處理，但這還需要再配合其他過濾器工具。Unix/Linux 系統上常用的搜尋工具有 grep、egrep 與 fgrep，以下分別介紹之。

11-6-1 grep 搜尋工具

grep（Global Regular Expression Print）的命令格式為：

```
$ grep [options] PATTERN [FILE...]
$ grep [options] [-e PATTERN | -f FILE] [FILE...]
```

其中常用選項（options）有：

- ✧ -c ：計算出行數。
- ✧ -l ：顯示出相符的檔名。
- ✧ -n ：顯示出相符的行號。
- ✧ -i ：忽略大小寫。

(A) 【純文字搜尋】

在檔案中尋找某一純文字字串，搜尋成功再印出該行，操作範例如下：

```
$ cat file_2                          【觀察 file_2 內容】
This is my question.
Is this a dog or cat?
$ grep my file_2                      【在 file_2 內搜尋 my 字串】
This is my question.                  【找到則印出該行內容】
$ grep 'dog or cat' file_2      【在 file_2 中搜尋 'dog or cat' 字串】
Is this a dog or cat?
```

(B) 【純文字附選項搜尋】

為了增加搜尋效益，可選擇附加某些選項，操作範例如下：

```
$ cat file1                          【觀察 file1 內容】
Is this a book?
Yes, this is a book
$ cat file2                          【觀察 file2 內容】
Where is my book?
$ grep -i T file1        【由 file1 中找出有 T 的字樣，而不論其大小寫】
Is this a book?
Yes, this is a book
$ grep -l 'Where' file1 file2  【找出 Where 字串是否在 file2 與 file2】
file2
$ grep -li where file1 file2       【同上，但不理會大小寫】
file2
$ grep -n 'I' file1        【在 file1 中找出有 I 字串的行號】
1:Is this a book?
```

(C)【正規表示式搜尋】

利用正規表示式搜尋，可以找出變化性較高的字串，以下介紹幾個範例：

- 『.』表示式：符合任何單一字元，操作範例如下：

```
$ cat file1
Is this a book?
Yes, this is a book
$ grep 'I.' file1              【找出 I 後面緊接著一個單一字元的字串】
Is this a book?
```

- 『*』表示式：符合空白或任何長度的字元，操作範例如下：

```
$ grep 'Ye*' file1        【找出前面是 Ye 的任何字串或字串】
Yes, this is a book
```

- 『[]』表示式：符合任何中括號內的字元，操作範例如下：

```
$ cat file_2
This is my question.
Is this a dog or cat?
$ grep [Mm] file_2          【找出有包含有 M 或 m 字串】
This is my question.
```

- 『^』表示式：一行的開始符合某一字串，操作範例如下：

```
$ grep ^This file_2          【找出行的開頭是 This 的字串】
This is my question.
```

➢ **『$』表示式：**一行的結尾符合某一字串，操作範例如下：

```
$ grep ?$ file_2             【找出行的結尾是 ? 的字串】
Is this a dog or cat?
```

習 題

1. 請簡述『外殼』(Shell) 與『核心』(Kernel) 兩者之間的關聯？

2. 為何稱 Unix/Linux 為開放性系統？這與 shell 又有何關聯？

3. 請簡述『外殼環境』(Shell environment) 的運作程序為何？

4. 在 Unix 系統之下，其標準輸入、標準輸出、以及標準錯誤輸出為何？應如何改變其輸出/輸入方向 (或位置) ？

5. 何謂『管道』(pipe，|) ？請舉例說明之？

6. 何謂『背景程式』(background)，應如何下達 (以 vi 為例) ？

7. 如何觀察出目前自己所執行有哪些程序？

8. 如何觀察出系統目前有哪些程序正在執行中？

9. 如何強制刪除正在執行中的程序？

10. 請說明 $ps –ef | grep ftpd 命令的功能為何？

11. 請說明輸入轉向 (<)、輸出轉向 (>) 與輸出附加轉向 (>>) 的功能為何？

12. 請舉例說明，如何在同一命令行內輸入多個命令？

13. 請說明『正規表示式』(Regular expression) 功能為何？

14. 請說明正規表示中 *、^ 與 $ 所表示的意義為何？