UNIX 初學者使用手册

A Quick Guide for unix Newbies

Edition 2.4E
September 4, 1995

This booklet was prepared with LaTeX2ε using the CJK package and TTF2PK for PK font generation from Chinese True Type Font.

The LaTeX2ε macro package CJK was written by Werner Lemberg (a7621gac@awiuni11.bitnet).

The TTF2PK program was written and prepared by 王佑中、林耀仁 and 李育叡，三位均來自國立台灣大學。

Copyright ©1995 楊景翔

本書作者允許任何人以各種方式自由拷貝、散佈本書之全部內容，唯上述之行爲均不得涉及商業利益，且本版權聲明頁必須存在每份拷貝之中。其它未及詳述之版權聲明應以自由軟體基金會 (Free Software Foundation) 之 GPL (General Public License) 爲依循根據。

For Victoria

序

一開始的時候，不過是厭煩於同事們不斷問些重複的問題，而想堵住他們的嘴，省得我的工作老是被打擾，所以就有了一份我稱之爲"local guide"的使用手册第一版(Edition 1.0)，主要是要讓同事們熟悉實驗室的工作站環境。由於時間緊迫，其中內容較偏重"介紹"性質，內容也都針對筆者的工作環境做說明，寫成之後只在實驗室內做內部流通，沒想過要對外公開。後來，覺得這樣的手册需要的人一定很多，又自認第一版的內容尚未能對新手有所幫助，只能說適度解決了一些基本的操作問題，心想應該可以幫助更多人，就有了動筆的念頭。斷斷續續寫了幾個月，在經過排版系統轉換、以及一些朋友的校閱、提供意見後，正式以"2.4 版"公開在網路，時間是 1994 年 8 月。

一年來，陸陸續續接到一些朋友來信指正，以及鼓勵，在此先致上由衷的謝意。一向拙於整理檔案，無法將這些朋友的大名列上來，如果眞的在意，請不要客氣，儘快和我聯絡。另外，一直想找機會把內容再做修正與增減，無奈工作繁忙，只能偷空東補西填，現在總算有點東西，我將它稱爲"Edition 2.4E"。前一版有關網路的部分已經刪除，一來減少篇幅，二來，TANet 上現有的電子資料已經很多，有需要的讀者可以自行取得，或參考 2.4 版的第 6 章做初步的認識。本書有關 X Window 的部分在以後的版本中也可能刪除。

這還是一本寫給"*UNIX* 初學者"看的書。主要目的仍在提供一個入門的參考來學習 *UNIX*，希望能對初學 *UNIX* 的人有所幫助。你可以直接拷貝本書或透過"ftp"取得（如果你不知道怎麼使用"ftp"，請你先請教別人）：

新、舊版本：　ftp://moers2.edu.tw/chinese-pub/documents/UNIX-Guide
新版本：　ftp://ftp.csie.nctu.edu.tw/Documents/UNIX-Guide
舊版本：　ftp://ftp.cis.nctu.edu.tw/Documents/UNIX-Guide

你也可以將這本書拷貝給別人，就像你免費拿到它一樣。但，所有對本書的任何處理均不得涉及商業行爲，如果你有任何需要，歡迎直接與筆者本人聯絡。

本書在編輯後期，有賴幾位熱心的電通所 U 組同事細心校訂，在此特別說聲感謝。即使如此，仍不敢保證書中內容沒有錯誤，讀者如果發現任何可以改進的地方，歡迎來信(Email)告知，筆者的電子郵件信箱是：JS@v0sun11.ccl.itri.org.tw 。 最後，還是祝各位

Happy UNIXing !!

楊景翔 寫於 新竹／臺灣
1995 年 8 月

Contents

1 出發之前 **1**

1.1 Login 1

1.2 File System - A Short Trip 7

1.3 Logout 11

2 介紹 UNIX 常用的指令 **13**

2.1 隨便先看看 13

2.2 不必急著學會 15

2.3 檔案權限 25

2.4 再接再厲 27

2.5 魔數 40

2.6 快學完了 42

2.7 1752 年 9 月 44

3 UNIX 簡史 **47**

3.1 失敗的計劃 47

3.2 誕生 47

3.3 欣欣向榮 48

3.4 Richard M. Stallman 49

3.5 編輯器 51

3.6 你可能不知道 52

4 你的殼 C Shell **55**

4.1 一個簡單的 .cshrc 例子 . . . 55

4.2 預定變數(Predefined Variables) . . . 56

4.3 別名(alias)與歷史(history) . . . 61

4.3.1 Alias . . . 61

4.3.2 History . . . 62

4.3.3 健步如飛 . . . 62

4.4 環境變數(Environment Variables) . . . 71

5 看看幾個實際的例子 **77**

5.1 One Example - .cshrc . . . 77

5.2 One Example - .login . . . 82

5.3 One Example - .logout . . . 86

5.4 有些事情你最好先知道 . . . 86

6 Tour of Tools **91**

6.1 V-eye . . . 91

6.1.1 小心你的鍵盤 . . . 91

6.1.2 vi 的模式有兩種 . . . 92

6.1.3 眞的來玩玩 . . . 92

6.1.4 ed 模式下的 vi . . . 96

6.2 sed 和 Regular Expression . . . 98

6.2.1 Find and Replace . . . 98

6.2.2 匹配行首：^ . . . 99

6.2.3 匹配行末：$ . . . 99

6.2.4 匹配任一字元：. . . . 100

6.2.5 多種選擇的匹配法：[...] . . . 100

6.2.6 匹配零或多個字元：* . . . 101

6.2.7 非常奇怪的表示法 . . . 103
6.3 關於 sed 的其它用法 . . . 106
6.4 簡介 grep 指令 . . . 107
6.4.1 grep 的選項 . . . 108
6.5 簡介 cut 的用法 . . . 109
6.6 順便談談 paste . . . 111
6.7 還有 tr . . . 112

7 學寫 C Shell Script 115
7.1 你的第一個 Shell Script . . . 115
7.2 用 Shell Script 改檔名 . . . 119

8 X Window System 123
8.1 Open the Window . . . 123
8.2 X Resources . . . 125
8.3 A Little More . . . 127

9 先別急著走開 129
9.1 引介一些有名的程式 . . . 129
9.2 參考一下 . . . 132
9.3 背景說明 . . . 135
9.4 Finale . . . 135

Chapter 1

出發之前

要使用 *UNIX* 你得先要有個帳户 (account)，你必須向你的系統管理者申請，他/她會給你一個"使用者名稱"(username) 以及一個"密碼"(password)。密碼用來保護你的檔案，保護系統，保護一切的一切。不要忘了，*UNIX* 是多人使用的系統，除了你，還有許多人也用同一個系統，爲了區別身份，每個使用者都會有個 username，就像每個人都有個名字一樣； password 就更不用提了，想想你的金融提款卡就知道了！

1.1 Login

有了帳户，上機頭一件大事就是 login。這一節將介紹與 login 有關的步驟及相關檔案；你將了解 *UNIX* 如何識別你的身份、如何把你安置在系統中、如何給你一個"家"！特別先提醒你，*UNIX* 區分大小寫，而且幾乎所有的指令，程式都是小寫的！這一點跟 DOS 有很大的不同。另外 *UNIX* 裡的檔案名稱沒有什麼禁忌，而且檔案長度可以到 256 個字元，有些老的 *UNIX* 只能接受最多 14 個字元的檔案名稱，但這種系統已不多見。你愛取什麼名字就叫什麼名字，不必有什麼三八的限制（MS-DOS 裡面，檔名只能有 8 個字元， 副檔名最多只能有 3 個字元）。

login 後，系統會顯示某個（段）符號，例如：

```
%
或者
ccsun2 /1 >
```

這一個符號叫 "系統提示號"(System Prompt)。此時你已經在 *UNIX* 裡面了。更精確的說，是已經在一個 Shell 裡面了。Shell 就是 "殼" ，它擔任一個翻譯者的角色，你給的每個指令都由 Shell 負責將它解讀並做處理。

到這裡爲止，好像跟 PC 的 DOS 差不多？我們使用PC 的 DOS時也是在 ">"後面下指令。對！你可以這樣說，但， Shell 本身卻也是 *UNIX* 眾多"指令" （應該說是"程式"）中的一個，所以，你可以使用不同的 Shell 來擔任你的翻譯者，你可以使用 /bin/csh（C Shell），也可以用 /bin/sh（Bourne Shell），或者由網路上取得的tcsh、bash等等，甚至你也可以自己寫一個 Shell 。這一點和 DOS就不同了，在 *UNIX* 裡，你可以選擇你的翻譯人（Shell），而在你申請帳號後，你的系統管理人會先替你指定一個預設的 Shell，一般來說，如果你用的是 Sun工作站，那會是 C Shell

，總之，只要你 login 得了系統，你就有個"殼"就是了。

那麼，你如何知道你用的是哪一個 Shell 呢？你可以看一看這個檔案： /etc/passwd。(用"more /etc/passwd"指令)。它是長成這樣子的：

```
.....
.....
john:ACrBqjTb8JmxQ:149:200:John Cooley:/home/users/john:/bin/csh
cpy:,xcrggtKWJTUA:172:200:C. P. Y:/home/users/cpy:/bin/tcsh
math:po.isgMVOPEw:110:200:Guess Who?:/home/users/math:/bin/sh
smith:ouzoi1EiDuJuk:129:200:Bob Smith:/home/users/smith:/bin/csh
student1:ou..41EiDuJuk:159:300:Talking Head:/home/users/student1:/bin/csh
.....
.....
```

這個檔案每一行定義一個 user，以"："爲分隔符號，定義一個 user 的login name(username)、password、login 後的"家" (Home directory) 以及 login 後要率先執行的"東西"等等。

由左至右以"："分隔的七項，分別說明於下：

1. username　第一項是使用者 login 時的 username，一般來說，使用者只能接受系統管理人給的名字，不能隨意認你選，比如，在學校，大都會用學號來命名，沒什麼大道理，就是方便管理而已。除非你有什麼後門 (跟系統管理人是好朋友啦，什麼的...)，否則，這個項目是無法改變的，如果你覺得你的 username 很拙，很炮，實在不喜歡，對不起，你得忍受它，把你心目中想要的好名字用在下面說的第五項，全名 (full name)，比較實際。

2. password　第二項是使用者的密碼，經過特殊編碼之後的文字，基本上你可以假設沒有人可以從這些亂七八糟的字元還原成你在login 時所 key in 的 password。密碼是可以由你自己隨意更改的，但要切記，不要用太簡單的密碼，像直接使用 username 當作 password，或者用簡單的數字、英文姓名，像 1234567、Annie 、James ，都不是好的 password，甚至可以說是不好的 password。盡量複雜，如 a5g2s7ts、s$bb%d!&—，當然，前提是你自己要記得住，在你下次 login 時，都不要忘記。系統管理人最討厭的事之一就是三天兩頭有使用者跑來跟他說"password 忘記了"。

3. user id　第三項是使用者的"身分證字號"，就像 username 是你的姓名一樣，這個號碼也是唯一的；換言之，在你所使用的系統中，沒有其他人跟你一樣名字 (username)，也沒有人的"身分證字號"(USER ID) 跟你一樣，而事實上，在 *UNIX* 裡面，系統眞正用來區別身份的是這個 user id。你可以用"id"這個指令來看你的 user id 和底下說的 group id。

4. group id　第四項是使用者所屬的群體編號，就像每個學生都會隸屬於某個班級一樣，在*UNIX* 裡，每個使用者也會被分類到他所屬的團體，而每個這樣的團體都有個獨一無二的編號，就是 group id。這種把使用者再分類的作法是在讓使用者可以更方便管理他們的檔案，比如說，同屬一個 group 的人可以共用某些檔案，並拒絕其他 group 的人讀取這些檔案。每個使用者一定屬於至少一個 group，換言之，一個使用者可能同時屬於好幾個 group。就像一個學生可以是一年甲班的學生，也同時是吉他社社員一樣。

   你可以用"groups"這個指令來看你是屬於哪一個 group。這個指令會顯示出你所在的 group，可能一個，兩個，也可能沒有。爲什麼沒有呢？那是因爲系統

管理人沒有幫你分類的緣故[1]。你可以看一下 /etc/group 這個檔，它的樣子有點像現在講的 passwd 檔，但只有 4 欄，依序爲："團體名稱:密碼:團體編號:該團體成員1，成員2，..."。

這個檔案也是系統管理人設定的，所以說，你自己並不能決定你要加入或退出某個團體。

5. full name　第五項是使用者可以自己更改的項目，一般都是用使用者自己的姓名，以方便知道 username 與眞人之間的關係。另外是在你寄出電子郵件時，收信人可以知道寄信的人是誰，而不必從 username 中去猜測。當然，你可以隨意替自己取個響亮的名字，像 Albert Einstein 啦，Bill Clinton 啦，都可以。

6. home directory　第六項是定義你的所謂"家目錄"。這好比你家的户口名簿上除了要登錄你的身分證字號外，還要詳細記錄你家地址一樣。這個家目錄就像你被分配到的公寓一樣，只要你一 login 到 *UNIX* 裡，系統就自動把你"安置"到這個屬於你的地盤。這也是不能隨你改的，由系統管理人分配。

這個"家"的觀念，在 *UNIX* 裡很重要，因爲 *UNIX* 系統是一個所謂多人多工的系統，也就是說，同一個系統內可能有上百人同時在使用，而且，每個人都享有同樣的系統資源，爲了避免不必要的麻煩，每個人都會被分到一塊磁碟空間當作"家"，每個人只能在自己的"家"裡面爲所欲爲，不可以干擾到別人。 所以你的一切檔案，包括垃圾，都只能放在自己家裡。你有絕對的權力來管理你的家，把門關上或把門打開，格局也隨你。當然如果你夠慷慨，你也可以打開你的家門讓別人也進來胡攪。但一般情況下，個人的家目錄別人只能跳進去看看檔案，而不能做什麼事情。有的人基於某些理由，更是連家門都上鎖了，別人連瞄一眼都不行。

7. shell　第七項是定義該使用者在一 login 完之後，所要執行的東西，一般是定義一個使用者所用的 Shell 名稱，也就是指定指令的翻譯人，這一項你可以自己指定，在你申請帳號時系統管理人給的只是暫定的，換不換隨你。

以上介紹了 /etc/passwd 這個檔案，其中有三項可以由使用者自己選擇，即 password、full name、shell。

password 的更改是你一定要會，而且經常要做的，有的系統甚至強迫使用者每隔一段時間就要換一次密碼。更改密碼用下面這個指令：

```
% yppasswd
Changing NIS password for Jacky on roxy.
Old password:
New password:
Retype new password:
NIS entry changed on roxy
```

如果無效，則用：

```
% passwd
```

(爲什麼有兩個改密碼的指令呢？稍後再提)

它會要求你先輸入舊密碼，再連著輸入兩次新密碼（第二次是用來確認），以完成更改的動作。

[1] 沒幫你分類，那麼你在 /etc/group 這個檔中就看不到你的名字，但是你還是會有個 group id，系統就用這個號碼來當你的 group 名字，當你用"groups"這個指令時，就看到數字。

值得一提的是，你的密碼雖然可以很長，但是絕大多數的 *UNIX* 系統都只認得前 8 個字元，所以，"MtosaJKopq"與"MtosaJKo12"是等效的，超過 8 個字元的密碼只是爲了好記或練打字罷了。另外有些系統規定密碼至少要 6 個字元，或規定密碼之中必需要含一個或以上的數字且含 2 個以上的英文字母等等，這些規定都是用來減少密碼被破解的機會。由此也可以看出 *UNIX* 系統有多怕這種密碼被破解的事，尤其系統管理人的密碼，更是要小心。

full name 的更改是用：

```
% chfn
```

它會先顯示你現在的 full name ，再要求你輸入新的。至於，full name 可不可以用中文，要視系統而定，有些系統會因中文碼而產生不可預期的結果，在確定這件事之前，最好不要嘗試，先問問你的系統管理人再說。

shell的更改是用：

```
% chsh
```

它也會先顯示目前你用的 Shell，如 /bin/csh，然後要求你輸入新的 Shell名稱。要注意的是，這裡的 Shell 名稱必須給一個系統認可的名字，例如 /bin/csh，或 /bin/sh 等等。

至於哪些是系統認可的名字，就要看 /etc/shells 這個檔了，如果沒有這個檔案那系統只能接受 /bin/sh 及 /bin/csh 當作你的 default Shell。這是說，你就沒法將 /etc/passwd 裡面，你的那一行最後一項改成 /bin/sh 或 /bin/csh 以外的東西，這並不表示你就沒辦法用其他種 Shell，例如你自己做了一個 Shell 或者在網路上取得其他 Shell，你的系統管理人當然沒法知道這世界上又多了哪種殼，而且要把它放到 /etc/shells 這個檔去。所以，要求系統管理人隨時應使用者要求去更動 /etc/shells 事實上是不切實際的。這樣說來，剛剛說使用者可以自由選擇自己的殼就有點騙人了？當然不是，在這裡說用"chsh"來改變使用者的 default Shell，這個 default Shell 是使用者剛進 *UNIX* 時就用到的，就是說一進到 *UNIX* 就馬上進這個 Shell，所以叫做"default" Shell。進了這個 Shell 後，你還可以再進其他你自己喜歡的 Shell，並不受限制。所以即使你改不了 default Shell，那也不是什麼大不了的事，頂多在 log in 之後再進你想用的"殼"就可以了。至於，在無法改變 default Shell 的情況下，怎麼進（改用）你想要用的 Shell ，後面再談。

另外要提的是，如果你的機器上面 /etc/passwd 這個檔很小，只有短短幾行，而且也沒看到你的 username 在上面，那麼再試試用"ypcat passwd"指令看看，原因也是稍後再說。

逐項說明完 /etc/passwd 這個檔案中各欄位的意思，我們再回頭看前面的例子。假設你是"student1"那個 user，在該行的第一欄即是你的 username，第二行是編碼過的你的密碼，第三欄是 user id，第四欄是 group id，第五欄是你的全名，第六欄是你的"家"目錄，最後一欄是你一 login 後，系統要執行的東西。這裡，最後一欄是設定爲 /bin/csh ，所以啦，你 login 後就直接進 C Shell 了。反之，如果在 /etc/passwd 中該行的最後一欄設定是 /bin/sh ，那麼，你進 *UNIX* 系統後，就會直接進到 Bourne Shell 內。

/etc/passwd 這個檔只有系統管理者 (System Administrator，System Manager，root) 可以直接更改，但前面也提到，你仍然可以用一些指令來更動部分原有的設定，整理成下面的表格：

項目	用來更改原來設定的指令
密碼(password)	"yppasswd"，如果沒有這個指令，則用"passwd"
全名(full name)	"chfn"
殼(Shell)	"chsh"

藉由這些指令，你可以改變 /etc/passwd 中自己那一行的三個欄位。其他欄位的更動，只有系統管理人才能做，一般使用者無法更改。

好了，現在來說明一下，爲什麼會有兩個改密碼的指令（一個是"passwd"，另一個是"yppasswd"）。passwd 前面加個"yp"是"Yellow Pages"的縮寫，它是 Sun 這個公司所發明的一種系統管理方式。後來因爲"Yellow Pages"是英國某個單位的註册商標，就改名叫做"NIS"，是"Network Information Service"的縮寫。

簡單的說，你現在可以在很多部機器用同一個 username、同一個 password 來 login 進系統，改密碼時也只要在其中一部機器更改，不必每架機器都做同一個動作；而系統管理人也只要在一部機器上建立每位 user 的資料，如 /etc/passwd 這個檔，不必每一部機器都去做同樣事；可以讓使用者在隨便一架機器登錄進系統，不必受限於只能由某特定機器 login。這些都是 NIS 的功能。由於這個發明，系統的運作更有效率，管理上也更容易，所以現今的很多 *UNIX* 系統都採用它，也因此，原來在 NIS 發明前的改密碼的動作就有些不一樣了，在古早以前，每一部 *UNIX* 機器都有自己的使用人，也就是說，每一部機器都有個 /etc/passwd 檔，如果你要使用機器A，那你得在機器A的 /etc/passwd 上有名字，要使用機器B，得在機器B的 /etc/passwd 有名字，要改密碼時，在機器A上用："passwd"，改了機器A的，機器B的還沒更動呢！

現在有了 NIS，使用者的資料都集中放在所謂的 NIS Server 上頭，其他機器只是所謂的 client，沒有必要保留像 /etc/passwd 這樣的檔案，這些資料只要在必要時由 NIS Server 提供即可，所以你不一定可以在你所使用的機器上找到眞正的 /etc/passwd，也就是說你看到的 /etc/passwd 只是該機器自己使用的而已，而這架機器可能不是 NIS Server，所以你可能看到一個很簡單的 /etc/passwd，那裡面沒有你的 username，而你依然可以 login。

那你要問，使用者怎麼知道哪一架機器是 NIS Server，好去看看眞正的 /etc/passwd 長什麼樣子呢？如果你的系統有跑 NIS 的話，可以用"ypcat passwd"來得到一個完整的 passwd 檔，或者用" ypcat passwd | grep 你的 username"來得到你自己應該在 /etc/passwd 的那一行。

而舊的 passwd 指令，和新的 yppasswd 有何不同呢？簡單的說是，yppasswd 改的是 NIS 的資料，而 passwd 改的是你用的那架機器的 /etc/passwd 資料，如果在這架機器上面沒有你的資料，那表示你的資料應該在 NIS Server 那邊，系統會自動去更改 NIS 資料，這時，passwd 和 yppasswd 的作用就一樣了，反之，萬一在你所用的那架機器上 /etc/passwd 有你的資料，那 passwd 指令就改這個檔而已，不會去改 NIS 的資料，這時跟 yppasswd 的結果（改 NIS）就不同了。

無論如何，一般如果有跑 NIS，還是用 yppasswd 這個指令來改密碼。 而且，只要在其中一架可以 login 的機器改就可以了，萬一沒有 yppasswd 這個指令或者沒有 NIS，再用 passwd 也不遲。

好啦，上面這一段算是把 login 的動作講完了，如果大家還想更了解 NIS，或是其他的細節可以找找其他相關書籍，或是去問你的系統管理人。本書以 C Shell 爲說明對象，如果你不是用 C Shell 或 T C Shell，有少部分內容可能會有些許不對，請讀者自己注意；而在容易出錯的地方，筆者也儘可能先點出來。

回到前面的話題，說到 login 後的系統提示號，prompt。有人不喜歡既定的 prompt 樣子，想自己弄一個特別的，例如，改成

```
I Hate UNIX >
```

這當然是可以的，下個這樣的指令就可以了：

```
set prompt = 'I Hate UNIX > '
```

關於設定環境的其他事項，在這裡先略過不談，在 C Shell 那章再好好研究。

假設你的 default Shell 是 C Shell（/bin/csh），那麼，從你 login 開始你就脫離不了它。你

下的所有指令都跟它有關。*UNIX* 迷人的地方之一便是，她允許你用很簡單的方法重新設定你自己的環境。某甲的指令 "del ABC"可能是把 ABC 這個檔案殺掉，但某乙下同樣的指令"del ABC"可能是把 ABC 這個檔案印出來。總之，你可以替你自己量身訂製一個屬於自己的 *UNIX* 環境就是了。而前面提到， Shell 居於你和 *UNIX* 之間，充當翻譯人的角色，所以，要替自己訂製一個特別的環境，只要好好教育你的翻譯人就可以了，像剛剛提到的例子，某甲告訴他的翻譯人說，"del"表示"殺檔案"，所以當某甲"del ABC"時，他的翻譯（殼）就去執行"殺檔案"的動作；而某乙則是教育他的翻譯說，"del"是"印檔案"的意思，所以，當某乙"del ABC"時，做的卻是"印檔案"的動作。

所以，你只要依你的喜好，教育好你的殼，往後你就可以輕輕鬆鬆差使你的殼去作一些你想做的事，使個眼色就好了，而且這種教育工作只要做一次就好，除非你改變主意了，要它換個樣子，否則，這種教育的功夫都不必再做。

這個教育的功夫很重要吧！想必有個特別的檔案來作爲教案，對的，這個教案就是".cshrc"，在你的"家"(Home directory) 裡面，它用來設定你 C Shell 的相關環境。

另外值得一提的是，在 *UNIX* 裡面，像".cshrc"這種用來設定環境的檔案幾乎都是 以 "."開頭來命名，如 ".cshrc"，".login"，".xinitrc"，等等，而這些以 "." 開頭來命名的檔案是半隱藏檔，當你用"ls"來看檔案名稱時，必需用"-a"這個選項(option)才看得到。

一般來說，在你申請帳號時，你的系統管理人（System Administrator，以下簡稱 SA）都會幫你準備一個適用你所在電腦環境的基本".cshrc"檔。

現在先不管".cshrc"，因爲在後面我們有專門介紹它的章節。我們來把 login 的動作先講完。

可能有人已經發現，有些人一 login 進系統就會自動執行一些程式，比如說，直接進到 X Window （或是 Openwindows）裡面，或者螢幕會出現個"Good Morning"的字樣，而有的人卻不會？爲什麼呢？這是另一個檔案".login"的功能。

顧名思義，".login"就是在 login 時產生作用的。它緊接著".cshrc"之後"執行"。爲什麼說"執行"呢？因爲".login"事實上是一個 user 自己寫的"program"，一個以 C Shell 語言寫成的程式(我們叫它： C Shell Script)。每一次當你 login，".login"就執行一次，而且只有一次，除非你用特別的指令再去驚動它。

它跟".cshrc"不同的是，".cshrc"在每一次進 C Shell 時，都會被 "看" 一下，所以你每多進一次 C Shell（比如說，在 OpenWin 裡面又開了一個 comdtool，xterm 或者執行某個 C Shell Script），".cshrc"就多被 "看" 一次；而".login"卻不會。

有人大概還不清楚什麼叫做"進"Shell，進去一個殼？什麼意思呢？現在假設你已經 login 完，在 *UNIX* 裡面了，你可以在系統提示號後面下這個指令：

```
% /bin/csh
```

然後

```
% exit
```

下完後，你一定發現好像沒什麼事發生一樣。是沒什麼事發生，只不過你又進了一個 Shell，這就好像一個經過多層包裝的紙盒一樣，你剛 login 的時候是在最外層，然後如果你再下如上的第一個指令(/bin/csh)，那就又往裡頭鑽進一層，那一層的環境跟在最外層是完全一樣的，而你也不會感到任何差別，只不過在它外面，還包了一層殼；第二個指令是跳出所在的那一層，以這個例子來說，就是回到最外層。當然，你可以一直鑽進去，就是重複一直下指令"/bin/csh"，然後，你可以再用"exit"指令又回到最外層。不管你在哪一層，你都把它當作完全一樣，因爲，它們都是一個

殼，完全一樣的殼！所以即使你忘了你鑽進幾層，不知道該"exit"幾次，那也不要緊，留在那裡也無妨。

前面不是說，改不了 default Shell 也沒關係嗎？ default Shell 就是最外層的殼，你 login 後會先在這層殼，然後，你可以再進你喜歡的殼嘛，就待在那裡面不要出來，直到要離開 *UNIX* 爲止，這樣，就達到換殼的目的了！比如說，你在 /etc/passwd 的最後一項是被定成 /bin/sh，那你又習慣用一個叫 tcsh 的殼，因爲它有類似 DOSKEY 的功能，而偏偏你用"chsh"又沒辦法將 default Shell 從 /bin/sh 改成 /usr/local/bin/tcsh（假設 tcsh 是放在 /usr/local/bin 這個目錄下），因爲你的系統管理人並沒有把 /usr/local/bin/tcsh 放進 /etc/shells。沒關係，以後 login 進來，就鑽進 tcsh 內去，都不出來，一直到要 log out 爲止。怎麼鑽進去？就是：

```
% /usr/local/bin/tcsh
```

就好了！

你現在知道爲什麼那個東西要叫做 Shell，殼，了吧，眞的是一層一層的。很多初學者老是奇怪，殼，到底是什麼意義，現在應該比較清楚了。

再回到 login 的話題，前面說到有個叫".login"的檔案會在".cshrc"之後被執行。但它只會執行一次，就是剛 login 到系統的時候那一次；如果你再進一次殼，只會再讓".cshrc"跑一次，".login"不會因爲你多進了幾次殼就多被執行幾次。

所以，".login"就可以用來作一些只在"login"時要完成的事情，例如，提醒你今天是幾月幾日，開會時間，或者，跟你說早安等等。還有，前面提到的"login 後直接進視窗"的功能也可以藉由在" .login"檔案裡面加入幾行文字來達成。

整個 login 的程序完成後，系統便停在 System Prompt ，等著聽你的指揮。或者說，你就進到最外層的殼，準備下指令給這個殼，叫它去做事了。

關於 C Shell 的細節，以及".cshrc"、".login"怎麼寫，還有很多值得討論與學習，現在暫且先擱下，留待後面章節我們會用較大篇幅來學習[2]。

下一個小節，我們先來認識一下 *UNIX* 的環境，看看到底這樣一個系統裡面有什麼值得一看的東西。

1.2 File System - A Short Trip

現在我們來逛一逛 *UNIX* 系統。*UNIX* 的檔案系統其實條理分明，但初學者經常不明所以，雖說這不影響到普通的操作，但是，了解各個重要檔案所代表的意義卻能讓你用起來更得心應手。

UNIX 基本上是由一個核心（kernel）再加上各個大小功能不同的程式和檔案組合起來的。前面提到的 C Shell (/bin/csh) 及 chfn (/bin/chfn) 都只是個別的程式罷了。他們存在系統的某處，安靜地工作，通常你不會注意到他們，也不必理會他們，但是了解 *UNIX* 的檔案系統不但有趣而且對你以後的使用都非常有用。

UNIX 的整個檔案系統是長在"root"之下的。"root"是一般的說法，就路徑的表示法來說，

[2] 本書中是以 C Shell 來說明的，不同的殼有不同的"啓始"檔案，".cshrc"及".login"是 C Shell 這種殼的啓始設定檔，如果你用的 Shell 不是 /bin/csh，這裡說的可能不太適用於你，請你請教你的系統管理人相關的細節。

是"/" (這跟 DOS 不同的是，DOS 的根路徑是用"\"表示，而"\"在 *UNIX* 中另有特殊意義)。"/"，同時也是你系統的系統管理人的家 (你可以看看在 /etc/passwd 這個檔，root 所在的那一行的最後一欄)。在那之下，比較重要的檔案和目錄我們現在來看一下：

/bin 這是 binary 的縮寫。在傳統的 *UNIX* 系統中，這個目錄存放了使用者(user)會用到的指令(command)。正因爲如此，有的 *UNIX* 系統 (如 SunOS) 便把這個目錄裡的東西全都歸併到 /usr/bin 這個目錄下去，這樣一來，這個原來的目錄就空無一物了，而爲了使舊有的程式，或者習慣舊位置 (/bin) 的使用者不致產生不便，這個目錄名稱就還是留著，然後在上面貼個遷移啓事，公告說：這個目錄現在是空的，請到 /usr/bin 去找你要的東西。這種貼公告的方式在 *UNIX* 中非常好用且常見，它有個專有名詞，叫"link"，連結。如果你用 "ls -l /bin" 來看它，你可能會看到:

```
% ls -l /bin
lrwxrwxrwx  1 root            7 Jun  3 01:26 /bin -> usr/bin
```

那個"->"符號就表示"/bin"所在眞正的位置是，"/usr/bin"。link 在 *UNIX* 裡是很重要的功能，像這裡說的/bin這個例子，利用了連結，就可以達到"殊途同歸"的目的，不論你是習慣舊的檔案位置(/bin) 或者新的位置(/usr/bin)，你可以不改變自己既有的習慣而仍然可以用得順手，因爲這兩個目錄其實是同一個 (房子是同一棟，但是大門有兩個)。有關連結，後面的章節會再詳細介紹。

/dev 這是 device 的縮寫。這個目錄包含了所有 *UNIX* 的週邊元件。特別一提的是，*UNIX* 對待週邊元件、裝置就如同他們也是檔案一樣，所以就有 /dev 這個目錄，底下有各式各樣的名字，各對應到相關的周邊設備。例如，磁帶機是 /dev/rst0，硬碟機是 /dev/rsd0 等。除此之外，這個目錄還存在一個無底黑洞 /dev/null，有關它的偉大能耐我們在後面會看到。

/etc 這是 et cetera 的縮寫。這個目錄包含了系統管理上所需要的檔案和子目錄，是很重要的目錄。在這裡面的重要檔案包括前面費了不少唇舌的"passwd"、"shells"、系統啓動時要去執行的 "rc"、"rc.local"、設定網路機制的"inetd.conf"、設定印表機的"printcap"(BSD 系列 *UNIX* 才有)、關於電子郵件的"aliases"等等。一般使用者不必理會這些檔案以及他們工作的原理、細節，嚴格來說，這裡是屬於系統管理人的地盤。

/home 一般規劃用來當作各使用者的家(home directory)。但每個系統管理人有他們自己的考量，所以也不一定就將使用者的家建在這裡，因此這個目錄在你的系統裡不一定存在。

/lib 這是 library 的縮寫。用來存放各程式 (包括作業系統本身及使用者自己的程式) 要用到的程式館及檔案。和 /bin 一樣，它也常被搬家了，放到 /usr 下面，成了 /usr/lib ，當然，"link" (連結) 是不可少的，也就是說在這種情況下，/lib 和 /usr/lib 是同一個地方。

/lost+found 這個目錄下通常是空的。但當檔案系統發生問題，例如停電造成不正常停機，在機器重新啓動時，有些檔案找不到該放的地方，那些無法找到家的檔案便會放在這個目錄下，所以叫做"lost+found"，系統管理人會去處理這些無主孤兒，替它們歸位。

/tmp 這是 temporary 的縮寫。用來放置各不同程式執行時所產生的暫存檔。這是在整個 *UNIX* 系統中，少數幾個使用者可以存放檔案的公用地方，任何人都可以存取

這個目錄，所以有些使用者除了自己的家目錄外，還將這個目錄當成一個放檔案的地方，這是不對的。千萬不可以把它當作是私人的垃圾場，它是系統資源的一部份，一旦它被占滿了，很多東西都動不了。而且，一旦重新開機，這個目錄裡的檔案通常會被完全清除。當然，也有的系統在開機時不會去清這裡的檔案，但無論如何，把這個目錄當成可以存放私人檔案的地方總是危險的。

/vmunix　這是 virtual memory *UNIX* 的縮寫。是 *UNIX* 的核心(kernel)，它的地位就像汽機車的引擎，*UNIX* 啓動時的檔案。

/var　這是 various 的縮寫。這個目錄下有幾個子目錄用來存放系統執行時的資料，這些資料隨時在變動，有時需要系統管理人定期清除，以利系統的順利運行。 使用者比較要知道的是：

- /var/adm　存放系統訊息及使用者使用系統的會計帳目。
- /var/spool　存放包括列印、電子郵件等等的相關檔案，跟使用者息息相關的是 /var/spool/mail 這個目錄。在那之下存放了各使用者的信箱(mailbox)。
- /var/tmp　跟 /tmp 類似，是一個公用地盤，像 vi 這個編輯器就會把它當作緩衝(buffer)，有時當你用 vi 編輯一個很大的檔案時，可能會造成這個目錄過於龐大，導致所謂"system full"，表示沒有多餘的磁碟空間來讓 vi 動作。 跟 /tmp 不同的是，開機時它通常不會被自動清除（即使如此，使用者仍然必須小心使用這個目錄）。

/usr　這是 user 的縮寫。使用者（系統管理者也是一個使用者；有特權的使用者）會用到的指令及執行程式幾乎都存放在 /usr 之下的子目錄內。值得注意的子目錄或檔案有：

- /usr/bin　你用到的指令、程式幾乎都在此。像 ls、cat、cp、mkdir、rm 等等。它跟 /bin 在很多系統中是同一個地方，利用 link 連結在一起。
- /usr/include　存放了標準的"標頭檔"（header files）。寫 C 語言程式的使用者對此應不陌生。
- /usr/lib　系統的程式館。許多程式執行時會來這裡找零件。它跟 /lib 在很多系統中是同一個地方（連結）。
- /usr/5bin
- /usr/5lib
- /usr/5include　如果你的系統屬於柏克萊(BSD)系列，可能會有這三個目錄。這三個是另一套 /usr/bin、/usr/lib、/usr/include，不同的是，後者是屬於 System V 系列。(關於 *UNIX* 的血統，在第三章會有一番說明。)
- /usr/ucb　放有 BSD 系列的指令，如 lpr、vi、finger 等等。ucb 是 University of California at Berkeley 的縮寫。
- /usr/etc　存放了系統管理上會用到的程式。

/usr/dict	存放了系統字典。spell 這個拼字檢查程式會用它。另外一個後來被發現的用途是拿這個字典來猜別人的密碼。那又是另外一件事了。
/usr/man	跟 /usr/share/man 是同一個地方（連結），放有 *UNIX* 的線上使用手册(on line help)。

也許你要問，爲什麼 *UNIX* 上的基本公用程式不放在一個地方，要分散在 /bin 和 /usr/bin 兩個地方，library 也分成 /lib 和 /usr/lib 兩個，現在有些系統早已把兩個目錄合而爲一了，顯然它們是可以在一起的。爲什麼當初要分開呢？

UNIX 起步的時間遠在20幾年前，是在 PDP-11 上發展，當時的硬式磁碟機的技術並不像今天這樣好，在機器上只連接了兩個不太大的硬碟，其中一個的容量較小但讀寫速度較快，另一個容量較大但速度較慢。當時的 *UNIX* 設計者，把每個硬碟都視爲一個檔案系統，他們把較小、速度較快的那個用來放系統的核心(kernel)，以及其他必備的檔案，這樣一來，在系統啓動的時候才能有最好的反應；另一顆硬碟速度較慢，但容量較大，就用來放跟user有關的資料。這兩個硬碟就分別掛在系統的"/"及"/usr"這兩個目錄上。也就是說，"/"和"/usr"其實是代表不同的兩個硬碟的。

基於這個歷史因素，到後來即使硬碟技術進步到在一個硬碟可以擁有 GigaBytes 的容量，在一些機器上（例如 Sun），核心（kernel，就是 *UNIX* 啓動時的東西）所在的那個硬碟，我們還是會把它 partition 成"/"及"/usr"兩部份。然而，在有些系統，"/lib"和"/usr/lib"；"/bin"和"/usr/bin"其實早已"link"在一起，只差在多個名字罷了。有些 *UNIX* 則已經完全進化到沒有將硬碟 partition 成"/"及"/usr"。

現在大家看到的 /bin 和 /usr/bin；/lib 和 /usr/lib，就是這樣來的。

大致介紹完幾個 *UNIX* 的檔案、目錄，有些好奇寶寶會問爲什麼他可以在眾多機器 login 而且所存取的檔案都是同樣的，比如說， 從機器A和機器B login 後，家目錄一樣，看到的東西也都完全一樣，跟機器好像沒關係。還常常聽到別人在說什麼 NFS 啦，mount 啦，這些奇怪的名詞。那得談到 Sun 公司的發明： NFS 和 NIS(YP)。基本上，如前面一節所說，NIS(舊名 YP)讓你可以在不同的機器都可以登錄(login)。NFS（Network File System 的縮寫）則讓你可以在不同的機器上機都看到同樣的檔案，比如說你的家目錄是放在機器A的硬碟裡，而你從機器B登錄進系統，還是可以使用、存取存放在機器A硬碟裡的東西。這個跟傳統 PC 的使用觀念是很不一樣的。關於 NIS 及 NFS 的細節是另一個課題，在這裡不再做介紹，超出範圍太多了。

總之，你登錄進系統後，系統會自動跳到你的家目錄所在位置，等候你的命令。

好啦，對於"進入" *UNIX* 系統，你應該有一點點概念了。在下一章，我們先來看看大家最常用到的"指令"有哪些，詳細舉些例子說明。爲什麼要特別強調"指令"呢？因爲，在 *UNIX* 中，你常用的所謂"指令"，事實上絕大多數是一個一個大小程式，只有少部分是屬於殼的功能，不是一個獨立的程式，而是內建在殼裡面。舉例來說，"passwd"，"chsh"，"chfn"等，都是獨立的程式，它們各位在"/bin/passwd"、"/usr/bin/chsh"、"/usr/bin/chfn"。也就是說，你可以在 *UNIX* 檔案系統中找得到它們。而像， "set"、"source"、"alias"、"exit"等等"指令"，在 *UNIX* 的檔案系統裡面，並沒有一個對應的獨立程式，它們就是屬於"殼"的內建指令（但是不要忘了，殼的本身也是程式。比如，C Shell 就是"/bin/csh"，Bourne Shell 就是"/bin/sh"）。[3]

在進入下一章之前，先談一談如何跳出 *UNIX* 。

[3] 在本書中，爲了說明方便起見，並不特意區分這些獨立的程式與殼的內建指令，一律以"指令"稱之。

1.3 Logout

要離開 *UNIX* 系統 有很多種方法，但絕不是像 PC 一樣關掉電源。

絕對不可以關掉電源!!

最簡單的理由：系統裡面可能還有其他人或其他程式在跑，關掉電源將使得這些人、這些程式功虧一簣!

眞正要緊的原因是： *UNIX* 在跑的時候，有許多檔案的相關資料並沒有馬上寫回硬碟中，而是放在記憶體內，這些資料會在適當時機被寫回硬碟，沒有人知道是甚麼時候，如果隨便關機，這些資料便有可能丟掉，導致檔案系統的混亂，而這是比當機更令系統管理人頭大的事，根據筆者的經驗，用過 PC 的 *UNIX* 新手總是保有關機的"好"習慣，這些人通常很厲害，可以在別人制止之前找到電源開關，啪的一聲關掉電源 。這時，你會聽到一些人的慘叫聲...

最簡單的離開方法是：一直下"exit"這個指令，直到系統的登錄提示號(login:)出現爲止(表示你已經跳出系統了)。再強調一次，千萬不要關電源，那是系統管理人的事。

下一章就讓我們來看看一些常用的"指令"。

Bourne Shell(sh), C Shell(csh), Threaded/Tenex C Shell(tcsh),
Korn Shell(ksh), Public Domain Korn Shell(pdksh),
Bourne Again Shell(bash), Z Shell(zsh), Almquist Shell(ash),
Extensible Shell(es), Plan 9 RunCommand(rc), Steve's Shell(ssh)...
...在 *UNIX* 世界裡，你絕對不會是無"殼"蝸牛!

by JS *(Who?)*

Chapter 2

介紹 UNIX 常用的指令

以下先用一個表格整理出你可能常用的一些基本指令、他們的原意以及使用範例。有人覺得 *UNIX* 的指令太多，太難記，其實並不會的。只要掌握住原則，*UNIX* 的指令其實很好記。這個原則是：*UNIX* 的指令幾乎都是以下列形式出現：

指令名稱	—選項	受詞
(Command)	(options)	(arguments)

而大部分的花招都在那個"選項"上，它前面有個"-"號。而指令名稱、選項則幾乎都是英文動詞的縮寫。記住這個原則，並善用線上查詢(man 指令)就沒什麼難得了你了。

下面列的指令是一般使用者比較常用的，可以說只要"知道"這幾個指令，你就可以是一個極爲老練的 *UNIX* 使用者了。乍看之下，好像還蠻多的，不要緊張，沒人要你把他們通通一下子全記起來，只要看過、有個印象就可以了。沒有人能夠光看一些白紙黑字就能熟悉 *UNIX* 的。較常用的，就熟能生巧，閉著眼睛都能敲得出名字；較少用的、有一段長時間沒用的，就全忘光了，這是很自然的，所以說，熟悉 *UNIX* 的不二法門就是用、用、用。

偷偷告訴你， 有很多 *UNIX* 使用者，其實都是一招半式闖江湖，十個指令玩遍各種 *UNIX* ！他們可以，你也可以！當然啦，"man"這個指令，是一定要會的。下面會告訴你爲什麼。

爲了方便查閱，所以以表列方式整理，後面會再詳細一點介紹。表列的順序大致代表各個指令的使用頻率，不過，那也因人而異。又，爲了說明他們的名稱由來，表列說明部分仍採用英文。特別注意黑體字的部分，它指出指令的原意，可以讓你不必再爲指令名稱煩惱。

2.1 隨便先看看

指令	說明	使用範例
ls	**l**ist	`ls -la ~`
cd	**c**hange **d**irectory	`cd ../../dir1`
pwd	**p**resent/**p**rint **w**orking **d**irectory	`pwd`
date	**date**	`date`
who	**who** is logged in on the system ?	`who`
whoami	**who am i** ?	`whoami`
more	page through a text file	`more ~/.cshrc`

cp	**cp**y files	`cp file1 file2`
cat	con**cat**enate files	`cat f1 f2 > file3`
man	**man**ual pages/On line help	`man man`
alias	**alias** a command	`alias ll ls -la`
history	display past commands	`history`
setenv	**set env**ironment variables	`setenv DISPLAY v0sun2:0`
set	what is **set** in this C Shell ?	`set`
stty	**s**et **t**erminal **ty**pe	`stty erase ^H`
lpr	print a file to printers	`lpr ~/.login`
lpq	printer queue status	`lpq`
mv	**mo**ve files around	`mv ~/.cshrc ~/kk`
rm	**rem**ove/delete files	`rm -r ~/.junk`
mkdir	**mk**e **dir**ectory	`mkdir dir1 dir2 dir3`
rmdir	**rem**ove **dir**ectory	`rmdir ~/dir1`
mail	**mail**	`mail john < letter`
exit	**exit** a shell or the system	`exit`
chmod	**ch**ange a file or dir's **mod**e	`chmod 755 batch`
which	**which** one	`which openwin`
vi	**vi**sual editor	`vi ~/.cshrc`
ps	**p**rocess **s**tatus	`ps -aux | more`
kill	**kill** a process or job	`kill 898 ;kill %2`
grep	**g**lobal **r**egular **e**xpression **p**rint	`grep "set" ~/.cshrc`
df	**d**isk space in **f**ile system	`df | more`
du	**d**isk **u**sage	`du ~/my-secret-dir`
telnet	connect to another system	`telnet 140.111.1.1`
rlogin	**r**emote **login** to another system	`rlogin v0sun2`
rsh	**r**emote excuting a **sh**ell command	`rsh v0sun2`
find	**find** files	`find dir1 -name file1 -print`
compress	**compress** files	`compress big-files`
uncompress	**uncompress** files	`uncompress File.Z`
uuencode	**U**NIX-to-**U**NIX **encode**	`uuencode f1 label > f1.uu`
uudecode	**U**NIX-to-**U**NIX **decode**	`uudecode f1.uu`
w	**w**ho is logged in ?	`w`
su	**s**witch **u**ser	`su v10xyz`
wc	**w**ord **c**ounting	`wc -l ~/.cshrc`
env	display all the **env**ironment variables	`env`
diff	What's the **diff**erence between 2 files	`diff file1 file2`
cmp	**com**pare 2 files	`cmp file1 file2`
tar	**t**ape **ar**chive	`tar xvf tar-file.tar`
head	the first n lines of a file	`head -50 this-file`
tail	the last n lines of a file	`tail -50 that-file`
cut	**cut** down specified fields of a file	`cut -c1-3 f1`
paste	**paste** together 2 files line by line	`paste f1 f2 > total.cmp`
split	**split** a file into pieces	`split file1`
tr	**tr**slate characters	`tr 'a-z' 'A-Z' f1 > f2`
sort	**sort** data	`sort address-book`
cc	**C** language **C**ompiler	`cc -g -o test test.c`
whereis	**where is** the source	`whereis cat`
nslookup	**n**ame server **look up**	`nslookup umc.com.tw`
tee	**T**	`make -f mfile |tee log`
file	What's a **file**'s type	`file /usr/bin/*`
touch	update a file's date/time	`touch file2`
ln	**l**in**k**	`ln -s ~/f1 ./f2`
unlink	**unlink** an existing link	`unlink ./example`

expand	**expand** TAB to spaces	`expand tab_file > f2`
uname	**u**nix **name**	`uname -a`
expr	evaluate **expr**essions	`expr 1 + 2`

2.2 不必急著學會

下面大致介紹各指令的用法，只介紹最普通常用的選項(option)，各指令的其他選項用法請讀者在有需要時，自行 man 一 man 該指令，不要忘了，唯有不斷的使用與嘗試才是熟悉一樣語言、機器的捷徑。

給初學者的建議是，先大概看過一次，有個印象即可，碰到要用的時候再回頭看，會更清楚。或是你也可以拿著這本書，一邊上機一邊玩玩，也不失爲一個好方法。但是，千萬不要想一次全部學會，學習一旦有了負擔，就不好玩了[1]。

ls　各位如果已經用過 DOS ，應該還記得剛開始學會的第一道指令： dir (或 DIR)。用 dir 做什麼呢？用它來列出磁碟上的檔案目錄。在 *UNIX* 裡面，相對於 dir 的，就叫"ls"，是 list 的縮寫。例如，你想看看自己的目錄有哪些檔案，你可以下這個指令："ls"。ls 最常用的選項是"a"、"l"和"F"。"a"是"all"的意思，表示把隱藏檔 (檔名以"."開頭的) 也一起列出來；"l"是"long"的意思，是把檔案的其他資料如檔案長度，上次修改日期等等列出來；"F"則是將檔案的型態在檔名的後面加上下列符號以資區別：*表示可執行檔，/表示目錄，@表示連結。

UNIX 指令的選項可以分開、調換位置，也可以合在一起，所以"ls -laF"、"ls -aFl"與"ls -l -a -F" 是完全一樣的。例如："ls -la /usr/bin"就是列出"/usr/bin"這個目錄下的所有檔案名字。

cd　change directory 的縮寫。就是變換目錄的指令，和 DOS 相同。要注意的是，它後面應該要有一個目的地目錄，例如你要改變目錄到"/tmp"，應該用這個指令："cd /tmp"。如果你在 cd 後面沒有給目的地，只給指令"cd"，則表示目的地是家目錄。這一點與 DOS 不同，在 DOS 裡面，cd 是印出目前所在目錄的意思。(在 *UNIX* ，要印出目前所在目錄，是用"pwd"這個指令。)

在 *UNIX* 中，有三個跟路徑有關的"符號"需要在一開始就認識清楚：

. "."在 *UNIX* 中表示"目前的路徑位置"，也就是你用"pwd"指令看到的那一串東西。所以它是隨時在改變的，你走到哪裡，". "所代表的值就跟著改變。

.. ".."在 *UNIX* 中表示"目前的路徑的上一層"，或稱"母目錄"。例如你現在是在"/usr/lib"這個路徑下，那麼，"cd ../man"表示你要"跳到 /usr 底下的子目錄，man"，換句話說，就是 cd 到 /usr/man 的意思。

~ "~"在 *UNIX* 中表示"家目錄"。所以"cd ~/mail" 就是跳到家目錄之下的"mail" 這個子目錄。而另外一個常用的用法是在 username 之前加上~，表示該 user 的家目錄，例如，"~john"是指"john"這個 user 的家目錄。而如果在"~"之後沒給 username 的話，就是指自己的 home directory 的意思。

date　顯示出今天的日期及目前時刻。平常的用法就是：

[1] 先警告你，這一節的份量有點兒多，如果看煩了，請先休息一下，翻到下一章看看"優你克思"的歷史。

```
% date
Mon May  8 19:32:19 CST 1995
```

注意，有時候系統的時間跟眞正的時間是不一致的，也不曉得是什麼原因，*UNIX* 機器上的時鐘總是不一定準確，每隔一段時間系統管理者就得重新設定它，以免和眞正時間愈差愈遠。

另外，你會發現在西元年度之前，有三個字母（以上例來說，就是"CST"），這三個字母是代表時區，比如說，格林威治標準時間（Greenwich Mean Time）的代號是"GMT"；台灣所在的時區是中原標準時區。各位用的如果是 SUN 工作站，在安裝時如果依照正確的時區選擇，選擇了台灣所在的時區，那這三個英文字母應該是，"CST"。巧的是，美國的中部標準時間的縮寫也是"CST"，在平常的使用上，這個時區的用途不大，但是電子郵件系統(Email)會在每封電子郵件上附上這個時間，所以，可能發生的情況是，當你發出一封信給在美國的朋友，他收到時若以該封信上記錄的時間來看，他會發現，竟然收到來自未來的 mail！當然，一般人是不會這樣認爲的。而就有幾次，當筆者在News 上 post 一些 article 後，收到一個位在美國的某家公司的回覆，該公司的News 系統的管理人跟筆者說，我送出的 news mail 上的時間，因爲無法讓該公司的 news server 處理（該 server 必需定期清除過期的 article，而對於未來的時間，當然完全沒概念）所以造成一些困擾，事情發生幾次後，筆者曾詢問過台灣 SunSoft，沒得到有用的回覆，後來因爲也沒再收到類似的抱怨，也就不管它了。至於，"CST"到底該換成什麼，或者就是"CST"沒錯，台灣所在時區的縮寫到底應該是什麼，更沒去理會了，如果讀者們知道這個問題的答案，麻煩你告訴筆者一聲。（在台灣，HP 的機器上， 時區應該都是設爲 "EAT"）

who 它告訴你現在你用的這個系統中還有哪些使用者。下面是一個例子：

```
% who
Jacky    console May 10 10:19
John     ttyp0   May 10 10:19
John     ttyp1   May 10 11:09
s20cjs   ttyp6   May 11 16:12   (140.113.68.150:0.)
root     ttyp9   May 11 10:17   (roxy)
```

在上面的例子中，每壹行各代表一個被佔用的終端設備。以上面的 5 行輸出爲例：第一行表示"Jacky"這個使用者正在使用"/dev/console"這個終端設備，時間從5 月 10 日 10 點 19 分起，一直到你下"who"的時候爲止還是。第二行及第三行表示"John"這個使用者於 5 月 10 日 10 點 19 分起開始使用"/dev/ttyp0"這個終端設備；另外又於 5 月 10 日 11 點 09 分起開始使用"/dev/ttyp1"這個終端設備。第四行表示"s20cjs"這個使用者在 5 月 11 日 16 點 12 分起，從"140.113.68.150"這個 IP 位址，透過電腦網路，佔用了"/dev/ttyp6"這個終端設備。第五行表示"root"在 5 月 11 日 10 點 17 分起，從名爲"roxy"的這架機器，透過電腦網路，佔用了"/dev/ttyp9"這個終端設備。注意"who"的輸出中已經把"/dev"省略，只把終端設備的名字，如"ttyp0"顯示出來。

上面所說的"終端設備"其實是個很抽象的東西，從這裡開始，我們用"終端機"來稱呼它。以下用比較淺顯的文字來說明。

前面說過，*UNIX* 是一個多人多工的系統，在同一時間可能有多個使用者在使用，即使只有一個使用者，該使用者也可能同時在做很多事。比如說，某 *UNIX* 機器上使用者A在編輯檔案，使用者B在跑一個程式，同時還在列印檔案。像這樣，一個系統同時給這麼多工作給佔住了，系統

怎麼把每個工作的結果顯示給個別的使用者看呢？例如，使用者A跟使用者B同時在"ls"他們的檔案，系統必需給使用者A和使用者B個別的回答，這些回答都得顯示給個別的"終端設備"，讓各使用者都能看到個別的結果。

所以，在每一次有使用者 login 到系統時，系統就給每個人一個終端機，每個終端機給個編號，然後，系統就知道什麼人給的什麼指令是從哪個終端機來的，而其結果需要送回到哪個終端機去。這樣一來，每個使用者，或者，同一個使用者使用的不同終端機，就不會互相干擾了。

你一定好奇，*UNIX* 系統裡面這些終端機應該要很多，隨時等著有人來用，而當有人使用時，系統便從某個地方找出一個還沒被用到的給這個新來的使用。對的，這些終端機就放在 /dev 底下，你可以"ls -l /dev/tty*"來看看有多少。而 /etc/ttytab 這個檔案則規劃這些終端機怎麼對應到各個使用場合。底下是一個簡化的 /etc/ttytab 檔案，取自 Sun 工作站所使用的 *UNIX* (SunOS)：

```
# name  getty                           type            status  comments
#
console "/usr/etc/getty cons8"          sun             on local secure
ttya    "/usr/etc/getty std.9600"       unknown         off local secure
ttyb    "/usr/etc/getty std.9600"       unknown         off local secure
tty00   "/usr/etc/getty std.9600"       unknown         off local secure
tty01   "/usr/etc/getty std.9600"       unknown         off local secure
tty02   "/usr/etc/getty std.9600"       unknown         off local secure
tty03   "/usr/etc/getty std.9600"       unknown         off local secure
                .....................................
tty0e   "/usr/etc/getty std.9600"       unknown         off local secure
tty0f   "/usr/etc/getty std.9600"       unknown         off local secure
                .....................................
ttyp0   none                            network         off secure
ttyp1   none                            network         off secure
ttyp2   none                            network         off secure
ttyp3   none                            network         off secure
ttyp4   none                            network         off secure
                .....................................
ttyTd   none                            network         off secure
ttyTe   none                            network         off secure
ttyTf   none                            network         off secure
```

你會發現，最左邊一列叫"ttyp??"，tty 是"terminal type"的縮寫，而最後那個"p"則是"pseudo"的意思，表示不是眞正的終端機，是一個虛擬的終端機，一般是用來支援網路的 login，這可能是最常用到的，因爲目前大部分的系統都有網路功能，使用者大都透過網路來使用 *UNIX* 系統。所以當你用"who"指令時，看到幾個 ttyp 是常有的事，後面跟的一個流水號，並不代表什麼先後順序。你可能又發現，怎麼有 ttyq 、ttyr 等等，甚至其他的，並不是"pseudo"的"p"字頭，那大概是因爲只有 ttyp0 – ttypf不足以應付越來越多的網路需求吧。下次，當你要 login 到某個 *UNIX* 系統，卻出現"Out of ptys"的訊息時，你應該知道，那是因爲所有可用的虛擬終端機都被用光了的緣故，要解決這個問題，你可以用"w"指令來看看該系統的使用者中，誰佔用最多虛擬終端機，然後請他關掉幾個，空出虛擬終端機來讓你進去。

在眾多終端設備中， /dev/console 是最基本也是最特殊的一個。它通常代表連接主機的那個終端機（包括螢幕，鍵盤）。所以，當某一部 *UNIX* 機器只有透過網路被使用，沒有人在該機器所直接連接的終端機前上機使用的話，這個 /dev/console 就空下來了。另外，如果系統有什麼狀況要"報告"，比如說，磁碟滿了啦，它也會把訊息顯示到 console 上來。所以，有時候你會發現某架機器（的螢幕上）一直自動顯示一些訊息，那是因爲系統自己在使用 /dev/console 的緣故。在某些時候，系統要使用 console，可是偏偏 console 已經被占住了，那系統只好自己找地方用，如果你用過 Sun 工作站，進了 OpenWindows，沒有開個 console 專用視窗，你大概碰過螢幕上

的東西被某些訊息打亂掉，要 refresh 螢幕後，才能把螢幕弄乾淨。那是因爲你佔住了 console，而又沒留個門給系統，系統逼不得已，只好破門而入，硬是把它要報告的訊息擠進來了。所以，console 是不能隨便占住的，如果要用，也要記得留個門，怎麼留呢？如果是在 X Window 裡面，開個 xterm，用 "-C"的選項，例如："xterm -C"；在 Sun 的 Openwin 內，也可以"cmdtool -C"即可。如果不在視窗內呢？對不起，跟系統一起搶 console 用吧，一般來說，這種機會不多，即使有，也無害，只是不方便罷了。

現在，我們來玩個遊戲。以前面一個例子來說，John 這個使用者佔用了"/dev/ttyp0"，也就是說別人就無法再使用它，因爲後來的使用者會被分配到其它的終端機。這樣其實只說對了一半，你當然無法從別人佔用了的終端機中輸入指令、文字，但是你卻還可以將輸出丟到別人佔用了的終端機去。像前面的例子裡（假設你不是"John"），如果你用"ls -la > /dev/ttyp0"，你可以想像得到結果嗎？要不要也試試"/dev/console"呢[2] ？

還有一個奇怪的終端機名叫"/dev/null"，它像宇宙黑洞一樣，所有進到它裡面的東西都會不見，專門用來"消音"，在以後的章節中我們會看到應用的例子。

另外要提醒的是，不同的 *UNIX* 系統有不同的終端機處理方法和程序，這裡所講的是以 SunOS 爲主，如果你所使用的系統不是 SunOS ，在有些方面就可能會有稍許不同，但基本上觀念相通，如果你對上面的講解實在不太明白，那也不要緊，這些眞的不影響你用 *UNIX* ，一點兒也不！

whoami 顧名思義，它用來確認你是誰，印出使用該指令者的 username。你會問：自己當然知道自己的 username ，還用問嗎？是的，等到你有一天用到它時，自然就知道它的用處了，。如說，用"su"指令借用別人的 shell，又忘了身在何處時 ...。

把這個指令拆開也是另一個同義的指令："who am i"，或是 "who am I"。這兩個指令其實是"who"指令加上其選項"am i"或是"am I"，跟"whoami"比較起來，輸出的格式稍微不太一樣，會把你所佔用的終端機也顯示出來。

你會發現這是極少數不照規矩來的指令： option 居然如此口語化，甚至不用加"-"符號，而且，如果你用的是 SunOS 的話，會碰到更絕的事：不論你用"who am i"或是"who AM i"或是"who am I"，甚至用"who are you"、"who x y"，其結果都一樣。那是因爲，它只看"who"的後面是不是有跟兩個參數，是的話，也不管是不是"am i"，全當做一樣。

UNIX 有時候也是會馬馬虎虎的 ☺ 。

more 這個指令用來看檔案，使用頻率可能僅次於"cd"和"ls"。例如要看 file1 這個檔案的內容，就用"more file1"。當檔案超過一頁時，會在螢幕的左下角顯示"--More--"的字樣，並有一個百分比數字告訴你現在已經看了整個檔案的多少部分。這時候，敲一下空白鍵 <Space Bar> 可以看下一頁，或敲一下 Return 鍵，可以一次多顯示一行，一行一行慢慢看。中途不想看了，可以用 Control-C （按住Ctrl 鍵，再按C 鍵）來中斷。

如果檔案很大，而你要看的東西在檔案的很後面，例如，在大約第 100 行，你可以用"more +100 file1"，直接從第 100 行看起；或者，你只想從某個地方看起，你可以用"more +/key_word 檔案名稱"，直接從檔案中含有"key_word"字眼的地方看起，這時候，"more"會從含有"key_word"字眼的那一行的前兩行顯示起。

"more"還有很多功能，例如在 more 一個檔案時臨時決定要修改它，可以按 v 鍵

[2] 像這樣粗魯的搶用別人的終端機，很可能招來一些抱怨，行動之前，最好先打個招呼。事實上，你只要看看這些終端機（/dev/tty??）的檔案權限，就知道你爲什麼可以以上述方法搶別人的終端機用了。

進入 vi 直接編輯它。

另外有一個和"more"幾乎一樣的指令叫"page"，它跟"more"最大的不同在："page"在執行時，會先把螢幕上的東西先清掉[3]。

cp 拷貝檔案的指令。例如把 file1 拷貝一份成 file2 用:

```
% cp file1 file2
```

使用"cp"指令要注意一件事，上述指令中，如果"file2"已經存在，它會被"file1"蓋掉，因爲你沒有叫 *UNIX* 再詢問你。

UNIX 一向如此：永遠聽話，非必要時絕不開口。所以當你說"把 file1 拷貝一份成 file2 "的時候，它並不會確認 file2 的存在與否[4]，所以 *UNIX* 提供一個"再確認"的選項"-i" ：

```
% cp -i file1 file2
```

這樣一來，如果"file2"已經存在，系統會再問你 "overwrite file2 ?"，表示它發現"file2"已經存在，請你再確認是否要把原來的"file2"蓋掉。你必須回答"y"來把原來的"file2"蓋掉，或是回答"n"表示不要[5] 。

cp 另外有兩個 option 也很重要，"-r"跟"-p"。先看"-r"的使用：

```
% cp -r ~/dir1 ~/dir_tmp/test
```

這個例子用"-r"選項將一個目錄以及在它之下（不管幾層）的所有檔案、目錄等，拷貝到另外一個目錄裡頭去。在較新版的 DOS，這個動作叫 copytree，其實在 *UNIX* 早就有這個功能了。

一般的 cp 動作，新拷貝的檔案、目錄的日期都是設定爲當時的系統時間，跟原始檔案的日期不一樣。如果你想保留原始檔案的日期在新拷貝的檔案的話，就用"-p"這個選項：

```
% cp -p file1 file2
```

拷貝檔案也並不全是一對一的，有時候，你需要將一大堆檔案拷貝到另外一個目錄。舉個例子，要把 3 個檔案（file1、file2、file3）拷貝到 dir_dest 這個目錄下，就要用：

```
% cp file1 file2 file3 dir_dest
```

用慣 MS-DOS 的人，通常用"copy file1+file2 file3"來將 file1 與 file2 接（合併）起來，成爲一個新的檔案 file3 。在 *UNIX* 裡，"cp file1+file2 file3"意思是"把 file1+file2 這個檔案拷貝一份，叫 file3"（記得嗎？ *UNIX* 的檔案名稱可以含有一些奇怪的符號，雖然"+"號並不算太奇怪，但 *UNIX* 是把它當成檔名的一部分）。

在 *UNIX* 裡，要把許多檔案合併在一起，成一個檔案，不是用"cp file1+file2 file3"， 而是要用以下會介紹的"cat"這個指令。

[3] 在 unix 中，有個指令可以把螢幕先擦乾淨，這個指令叫： "clear"。

[4] 它假設主人永遠是對的，而實際的情況是，主人常常犯錯。

[5] 其實，只要你的回答是"y"開頭的字串，它都當做是肯定的答案，所以你回答"y"，"yes"，"ykk"都表示要將"file2" overwrite 掉。而像"n"、"NO"、"YES"，等回答，或是直接按下 RETURN 鍵，都是表示否定的意思。

UNIX 裡，有個很重要的觀念：所有的週邊裝置，其地位跟一般檔案是一樣的。而在所有的輸出入裝置中，哪個最重要呢？就是你用的鍵盤（用來讓你輸入）和螢幕（用來顯示／存放輸出結果）；在 *UNIX* 的術語中，他們就分別叫做：標準輸入裝置（standerd input），及標準輸出裝置（standard output）。有些指令會有所謂"destination"（輸出目的地），比如說前面說的"cp file1 file2"這個指令，file1 是所謂"source"（來源），而 file2 是所謂的"destination"（目的地）。

但是有很多指令是不必給"目的地"的，我們說這樣的指令有個"內定的輸出目的地"。比如前面說的 ls 這個指令， 你只要下"ls"或"ls 目錄名稱"，就可以看到它的輸出直接顯示在螢幕上，螢幕（標準輸出裝置）是它的"內定輸出目的地"，

大部分的 *UNIX* 指令，輸出目的地是可以隨你的意思更動的。改變輸出地，在 *UNIX* 裡我們叫它做"輸出重導向"（redirect）。例如剛剛的 ls 指令，你用它來列出該目錄下的檔案名稱，輸出到螢幕上。如果你要把輸出結果存起來，放到一個檔案裡呢，就要做"重導向"的動作（原先的輸出是到螢幕上，現在則是要輸出到一個檔案，輸出要被"重導向"到一個檔案）。重導向的動作由"＞"這個符號來完成。例如你要把"ls -la"的結果存到一個叫"list"的檔案：

```
% ls -la > list
```

這時你在螢幕上就看不到"ls -la"本來的輸出結果，因爲它們已經被放到"list"這個檔案裡了。如果是要將結果"附加"（append）到檔案，則是用"＞＞"：

```
% ls -la >> list
```

這時你在螢幕上也看不到"ls -la"本來的輸出結果，它們被加到"list"這個檔案裡去了。如果"list"是一個已經存在的檔案，"ls -la"的結果會附加在"list"原有內容的後面；如果"list"本來不存在，則系統自動產生它。

cat 這個指令不是"貓"的意思，而是 catenate（或 concatenate ）的縮寫。顧名思義，是把東西串起來、連接起來的意思。所以，"cat file1 file2"就是把 file1、file2 連接起來的意思。接起來以後呢？照例，輸出到螢幕上，因爲這個指令有"內定"輸出地–標準輸出（螢幕）。

所以當你下完"cat file1 file2"這個指令，你會看到這兩個檔案連接起來的結果出現在螢幕上，如果你是要將其結果輸出成一個檔案，則需將本來出現在螢幕的輸出結果"重導"到檔案：

```
% cat file1 file2 > file3
（相當於 DOS 的 copy file1+file2 file3）
```

你也可以只"cat"一個檔案，如"cat file1"，這樣的結果就像在 DOS 底下的"type file1"指令一樣。好啦，那再應用前面說的"重導向"，下面這兩個指令可以說有完全一樣的效果[6]：

```
        % cat file1 > file2
等於：  % cp file1 file2
```

在某些 *UNIX* 版本，像 SunOS，"cat"也能用來將檔案內容加上行號，用的是"-n"這個選項，如以下這個指令：

[6]永遠不要忘記：在 unix 的世界裏，總有不同的方法來完成同一件事。但是，也不要忘了盡可能選擇一個最簡單的方法！

```
% cat -n file1 > file1-with-line-numbers
```

是將 file1 這個檔案裏每一行文字最開頭加上一個號碼，第一行是"1"，第二行是"2"等，餘此類推。然後，將這個加了行號的內容，放到名爲"file1-with-line-numbers"的檔案中。

這個加上行號的功能其實也可以用"nl"這個指令來達到：

```
% nl -ba file1 > file1-with-line-numbers
```

使用"cat"指令最需要注意的是這個用法：

```
% cat file1 file2 > file1
% cat file1 file2 > file2
```

以第一個指令來說，你的原意是要增加 file1 的內容，就是在 file1 的最後面補上 file2，事實上，用 "cat file2 >> file1" 就可以達到目的。用"cat file1 file2 > file1"的結果是："cat file2 > file1"！另外附贈一個系統訊息："cat: input file1 is output"！

爲什麼呢？因爲，當你用到"＞"來重導輸出到一個檔案時，系統馬上先開啓一個空檔案來準備存放"＞"之前那個指令的輸出結果，所以在"cat file1 file2" 這個動作還沒有發生前，"＞"後面那個檔案"file1"就先被掏空了，整個結果變成是"連接一個空檔案和 file2，把結果放到 file1 中"！

第二個指令"cat file1 file2 > file2"，也是同樣道理，在"cat"還沒發生作用前，file2 就先被破壞掉了。

在使用"重導"時，千萬要特別注意。

man 這個指令是"manual"的縮寫。你會常用它來看某個指令的用法。例如，用"man ls"來看"ls"詳細的用法。提醒各位讀者，"man"這個指令一定要會，而且，一定會常用到。沒有哪個高竿的 *UNIX* 使用者是死背指令的，不常用的指令一定會忘記，我自己常保存一份檔案，記錄特別奇怪，或常會忘記的指令，碰到忘了的指令用法，就可以先看看這個檔案。時間久了，這個檔案就會變得很豐富，而不會用的指令也會愈來愈少。漸漸的，就沒有"指令用時方恨少"的懊惱了。

再回頭來看"man"的用法。你可以用"man -k 關鍵字" 來尋找跟"關鍵字"有關的指令[7]。

例如，你想知道有那些跟"copy"有關的指令：

```
% man -k copy
ttcp (1) - copy files in a ToolTalk-safe way.
bstring, bcopy, bcmp, bzero, ffs (3) - bit and byte string operations
copy_home (8) - fetch default startup files for new home directories
cp (1) - copy files
......
......
cpio (1) - copy file archives in and out
dd (1) - convert and copy files with various data formats
pg (1V) - page through a file on a soft-copy terminal
rcp (1C) - remote file copy
tcopy (1) - copy a magnetic tape
```

[7] 有的系統會有另一個跟"man -k"相當的指令叫"apropos"。

```
uucp, uulog, uuname (1C)    - system to system copy
uuto, uupick (1C) - public system-to-system file copy
......
......
```

先看看這個輸出結果，最左邊的就是跟關鍵字有關的指令，括號內的數字是表示該指令的 manpage 是屬於的哪個 section ；"-" 右邊是簡單的解釋。

manpage 是 manual page 的縮寫，指的是一篇篇描述各指令用法的特殊檔案，以 troff/nroff 語法寫成，長得像這樣子：

```
.\" @(#)cd.1 1.9 90/02/15 SMI; from UCB 4.1
.TH CD 1 "9 September 1987"
.SH NAME
cd \- change working directory
.SH SYNOPSIS
.B cd
[
.I directory
]
.SH DESCRIPTION
.IX  cd  ""  "\fLcd\fP \(em change directory"
.IX  "file system"  "cd command"  ""  "\fLcd\fP \(em change directory"
.IX  "working directory"  "cd command"  ""  "\fLcd\fP \(em change directory"
.IX  change  "working directory"
.IX  change  "directory"
.IX  directory  "change working"
.I directory
becomes the new working directory.  The process must
have execute (search) permission in
.IR directory .
If
.B cd
is used without arguments, it returns you
to your login directory.
.\".LP
.\"Because a new process is created to execute each command,
.\".B cd
.\"would be ineffective if it were written as a normal command.  It is therefore
.\"recognized and executed by the shells.
In
.BR csh (1)
you may specify a list of directories in which
.I directory
is to be sought as a subdirectory if it is not a subdirectory of the
current directory; see the description of the
.B cdpath
variable in
.BR csh (1).
.SH "SEE ALSO"
.BR csh (1),
.BR pwd (1),
.BR sh (1)
```

上面這個檔是系統中描述"cd"用法的 manpage ，（存在檔案系統的某處，在以後的章節會再談到）當你"man cd"時，系統先會找到這個檔，然後再用"nroff"這個程式去處理它，才變成你在螢幕上看到的樣子，這也是爲什麼你在使用"man"的時候，常會先看到"Reformatting page. Wait..."的字樣，然後眞正的內容才出現。

這些 manpage 存放的位置依指令的功能類別不同而不同，我們說它們屬於不同的"section"。傳統上， *UNIX* 的 manpage 依功能分類爲以下 8 個 sections ：

1. 使用者常用指令
2. 系統呼叫（System Calls）
3. 程式庫（Library Functions）
4. 週邊裝置及其驅動程式
5. 檔案格式
6. 遊戲程式（Games!!）
7. 其它
8. 系統管理上所需要的特別指令

上面我們用"man -k copy"的結果，發現"pg"也跟"copy"這個字眼有關，很好奇，想看看它到底是怎麼用，這時候，依照上面的指示，就用"man 1v pg"。這裡，在"man"後面的"1v"是告訴系統你要看的是 section 1 的 "pg"。爲什麼要指定 section 呢？因爲，同一個指令名稱可能會有好幾個 manpage 來描述。在沒有指定 section 的時候，系統自己幫你挑一個 section ，通常是先找 section 1，因爲它是存放一般使用者指令的地方 。所以，在一般的使用上，我們大都只是"man 名稱"，較少再指定 section 。你可以試試下面這些指令，看看結果各是什麼："man 1 time"；"man 2 time"；"man 3 time"；"man time"；"man 1"；"man p"；"man man"。你也可以比較"man core"和"man 5 core"看看兩者的速度差異。

你的系統很可能無法完全使用"man -k"這個功能，出現類似這樣的訊息："/usr/man/whatis: No such file or directory"，那是因爲你所使用的系統並沒有所謂的"whatis database"，可能是你的系統管理人忘了做"makewhatis"（或"mkwhatis"），或磁碟空間的限制等其它因素，無論如何，你可以請你的 SA 來看看什麼地方出問題。

setenv　這個指令是 set environment 的縮寫。用來設定你的環境變數，有關的細節在 C Shell 的章節會說明，這裡先略過。

set　這個指令跟 setenv 有幾分相似，也是用來設定 Shell 中的變數的，同樣的，它的細節在後面章節解釋".cshrc"和".login"這兩個檔案時再一併解釋。

stty　這個指令最常用來設定某些特殊控制字元在鍵盤上的定義，例如下面的例子設定了"Control-H"成"erase"的功能，也就是說按"Control-H"時，其動作是"back space"(倒退鍵)。

```
% stty erase ^H
% stty -a
 ........
erase  kill   werase   susp   intr   stop   eof
 ^H     ^U     ^W       ^Z     ^C    ^S/^Q  ^D
```

第二個指令是印出目前的設定，你可以先看看有那些是需要變更的。最常用的是上面列的幾個設定:其中"stop"的設定用來控制螢幕的捲動，很多初學者不小心按了"Control-S"，螢幕不動了，以爲當機；其實只要再按個"Control-Q"就好了。

lpr 這個指令是 line printer 的縮寫。古早以前，印表機並沒有現在進步，都是那種印起報表會嘰哩呱啦叫的 line printer。這個指令的名稱就沿用到今。常用的方式是："lpr -Plp1 file1"。其中"lp1"是印表機名稱，它是由你的系統管理人取的名字，定義在 /etc/printcap 這個檔案裡面，有興趣的人可以瞄幾眼[8]。如果你的系統有不只一部印表機，你得小心給對名字，以免將檔案印到遙遠的另一部印表機，徒增麻煩。如果你還不知道你該使用的印表機名稱，現在就去問你的系統管理人吧！

你會問：如果不給印表機名稱呢？一般來說，印表機名稱的內定值是"lp"，有的系統已經將其印表機取名成"lp"，那麼你用"lpr file1"跟 "lpr -Plp file1" 是一樣的。總之，不知道印表機名稱時，問問系統管理人就是了。

lpq/lprm 如果你想知道現在印表機工作的情形，看看剛剛印出去的東西到底有沒有在印，可以用"lpq"這個指令。它會顯示你的印出物的工作代碼(job number)，是一個系統給的流水號。如果你想取消列印，則用"lprm 工作代碼"。"工作代碼"是你用 lpq 指令看到的 job number。[9]

這裡舉個例子：

```
% lpr curves.ps
% lpq
Rank   Owner      Job  Files                 Total Size
1st    john       213  curves.ps             176320 bytes
% lprm 213
Bart: dfA660roland dequeued
```

mv 這個指令是 move 的縮寫。顧名思義就是用來將檔案搬來搬去。例如，將檔案 file1 搬到目錄 ABC 去：

```
% mv file1 ABC
（ABC 是已經存在的目錄名稱）
```

如果目錄 ABC 原先並不存在，那上面的指令就有不同的意思了，是說"把檔案 file1 變成一個新名字 ABC"，file1 就不見了，被搬成一個叫 ABC 的檔案了。這個效果跟 DOS 下的 rename 一樣："mv 舊檔名 新檔名"。要注意，如果 ABC 是一個已經存在的檔案，則 ABC 會被蓋掉，消失不見。跟先前說的 "cp -i" 一樣，使用 "mv" 的時候，應該要養成習慣加上 "-i" 這個選項："mv -i 舊檔名 新檔名"，萬一"新檔名"早已存在時，可以先讓系統再問問你的意思，免得把重要檔案弄丟了。

rm 這是 remove 的縮寫。顧名思義，就是把檔案清除掉，例如：

```
% rm file1 file2 file3
```

是把 file1 、 file2 、 file3 這三個檔案殺掉（在 DOS 中是用 del 這個指令來清除檔案）。千萬特別注意， *UNIX* 裡面是沒有 undelete 這回事的，一個檔案一旦被殺掉了，就救不回來了，動手殺檔案之前千萬三思！

和"mv -i"與"cp -i"一樣，通常在殺檔案前，我們要讓 *UNIX* 再問我們一次："rm -i file1 file2 file3" 。"i"這個選項是"inquery"的縮寫，表示詢問，也有人說是"interactive"的意思，表示與系統一來一往的對話、回答。如此一來，系統在動手殺每個檔案之前都會再問你一次，確認一下。你得一一回答"y"（yes的意思）才會真的殺掉該檔案，你也可以回答"n"（或按 RETURN 鍵）來反悔[10]。

[8] BSD 系列的 unix （如 SunOS）才有這個檔案，System V 系列的 unix （如 HP-UX）有另外的方法定義 printer。

[9] 如果你用的是 System V 系列的 unix （像 HP-UX），則用"lp"來印表；"lpstat"來知道印表機工作的情形；用"cancel 工作代碼"來取消列印。

[10] 請參閱前面在 "cp -i" 那部分講的，這裡也是同樣道理。

相對於"-i"這個選項的，是"-f"這個選項；他的意思是"force"，就是殺意堅強，不管三七二十一，不必再詢問的意思。這個選項殺傷力極強，非不得已不要使用！你可能要問，如果這兩個選項一起用呢？系統會詢問嗎？結果是：不一定，有的系統（像：SunOS）以"-i"爲優先，有的（像：HP-UX）則以"-f"爲優先[11]。

另一個很重要的選項是"-R"（在某些 *UNIX* 系統，你也可以用小寫的"-r"）。"R"是"Recursively"的意思，表示一層一層往下一路殺下去，不管有多少層子目錄。這就是在較新版的 MS-DOS 中的"deltree"這個指令。例如："rm -Rf junk"，會將"junk"這個目錄以下所有東西，包括檔案、目錄等，都殺掉，連"junk"目錄本身也會不見，又因爲有"f"選項，所以即使要被刪除的檔案有屬性（permission）的保護，也照殺不誤。

mkdir　make directory。熟悉 DOS 的人應不陌生。它是新造一個目錄的意思。例如："mkdir dir_ABC"，是在目前的路徑下產生一個叫"dir_ABC"的目錄。

rmdir　remove directory。熟悉 DOS 的人也不陌生。它是清除一個空目錄名稱的意思。例如："rmdir dir_ABC"，是將目前路徑下的"dir_ABC" 這個目錄清除掉。如果該目錄不是空的，也就是說裡頭有其它檔案或還有子目錄，那你會得到"Directory not empty"的訊息，告訴你，該目錄裡面還有東西，在那些東西還沒拿掉前，不能殺掉這個目錄。

mail　這是電子郵件的應用程式，如果你已經把信件先打好（假設檔名是"letter"），要寄給某人，他的郵件地址是 somebody@somewhere，則可以用：mail somebody@somewhere < letter。除此之外，筆者並不建議使用 mail 這個程式來收、發電子郵件，因爲它眞的不如其它著名的電子郵件程式好用[12]。

exit　這是跳出／結束 shell 的意思。如果你已經在最外層的 shell 那麼這個指令就與 logout 有同樣效果。

2.3 檔案權限

要解說下一個指令之前，先來談一個有關檔案的觀念：檔案權限(file permission)。

前面提過，*UNIX* 讓每個使用者可以在自己的家目錄裡管理自己的檔案系統，但是一個 *UNIX* 系統有那麼多人在使用，*UNIX* 是靠什麼來歸類，讓檔案系統不會亂掉呢？例如，使用者A的檔案如何不被使用者B殺掉？使用者B的信件如何不被使用者C偷看？使用者C寫的一個執行檔如何不被使用者D偷偷執行？解決這些問題，*UNIX* 的方法很簡單，就是將每個檔案、目錄，都標上他們主人的名字，並且要求他們的主人要自己標示清楚誰可以讀、誰可以修改、誰可以執行等等。對檔案的管理權完全下放給使用者本人，人性十足。

當你用"ls -l"指令查看檔案時，會顯示類似下列文字：

```
drwxr-xrw- 1 john project1 128 Apr  4 12:19  dir
-rw-r--r-- 1 john project1 545 Apr 14 12:19  file1
----r--rwx 1 john project1 124 Jan 13 09:01  file2
lrwxrwxrwx 1 john project1   6 Jul 22 09:22  file3 -> file4
-rw------- 1 john project1 212 Jul 22 14:23  file4
        ..............................
```

[11] 但是，話說回來，有必要這樣讓系統進退兩難嗎？

[12] 有關電子郵件及其它網路的細節，請參考前一版第六章，或其它電子文件。

左邊算起的各個欄位依次爲：檔案或目錄的存取權限；連結的個數或子目錄個數；該檔案、目錄的擁有者；擁有者所屬的 group 名；檔案大小（以 Byte 爲單位）；該檔案、目錄建立的日期、時間；最右邊一欄是檔案（或目錄）名字。

檔案（或目錄）的存取權限那一欄位又分爲十個小項（1+3+3+3），依次爲：檔案型態（1項）；檔案擁有者（user）對此檔案的權限（3項）；與檔案擁有者同一群體者（group）對此檔案的權限（3項）；其他人（others）對此檔案的權限（3項）。一個檔案的權限有三種： 可讀(readable)、可寫(writable)、可執行(excutable)。以上例的 file1 而言：

```
-rw-r--r-- 1 john project1 545 Apr 14 12:19  file1
^^^^^^^^^^
```

第一項是檔案型態，一般來說，你最常看到的型態有三種：普通檔案（以"-"來表示是普通的檔案）；目錄（以"d"來表示是目錄，directory）；連結（以"l"來表示是連結，link）。

接著的"rw-r--r--"以三個爲一組（3+3+3），前三個是"rw-"，這三個字元（character）表示擁有者自己對此檔案的權限是：可讀(readable)、可寫(writable)，非執行檔案(-)。

中間三個是"r--"，這三個字元表示與檔案擁有者同 group 的人，對此檔案的權限是可讀、不可寫、不可執行。

後三個是"r--"，這三個字元表示其他閒雜人等對此檔案的權限也是可讀、不可寫、不可執行。

藉由這種檔案權限的規範，整個 *UNIX* 的檔案系統才能有條不紊，而每個使用者的檔案也能夠得到適當的保護。前面提到的 group 是 *UNIX* 中另一個將使用者分類的方法，將某些使用者定義成同屬某個 group ，然後這些同一個 group 的人就可以藉由檔案權限的變化，達到檔案共享而又不致於讓不相干的人看到或執行的目的。所以，當你下次看到"Permission denied"這個訊息時，你該知道是什麼原因了。

在第一章我們說過， group 是由系統管理人定的，一般的使用者無法自己定 group，道理其實很簡單：如果允許使用者自己定 group，使用者豈不是可以三不五時更改自己的 group，侵入別人的 group 來"偷"東西嗎？

好啦，現在你可以試著解釋上面例子中 file2 的權限是什麼。還有，你覺得這樣的檔案權限合理嗎？

接著再看上面一個例子的第一行：

```
drwxr-xrw- 1 john project1 128 Apr  4 12:19  dir
```

第一個字元"d"表示它是一個目錄，它的權限劃分和前面說的都一樣，只有一點要特別注意，就是"執行"權限那一項。如果它被設爲"不可執行"，那麼這個目錄就"跳不進去"。怎麼說呢？以上面的例子來講，"dir"這個目錄的權限是"rwxr-xrw-"，依照我們在前面所學到的，表示這個目錄的擁有者可讀、可寫、可執行這個目錄，對一個目錄來說，所謂的"可執行"，指的是"可進入"。所以，這個例子是表示這個目錄的擁有者和同 group 的人都可以跳進這個目錄內，但是其他人就不行。雖然"其他人權限"的部分是設爲"可讀、可寫"，但因爲是"不可執行"，所以就是說，這個目錄不對"其他人"開放的，非同一 group 的人如果要"cd dir"，只會得到"permission denied"的回答。

至於私有檔案的權限，檔案擁有者本人當然可以全權處理。要改變一個檔案的權限，用等一下要介紹的指令（chmod）。

最後來看檔案型態是"連結"（link）的例子。

```
lrwxrwxrwx 1 john project1   6 Jul 22 09:22  file3 -> file4
-rw------- 1 john project1 212 Jul 22 14:23  file4
```

看上例中的 file3 ，它的檔案型態是"l"，表示它是一個連結的檔案，只是一個"門牌"而已，眞正的內容是在 file4。注意 file3 的權限，是"rwxrwxrwx"，也就是所有人都是"可讀、可寫、可執行"。但是，再看看"門牌"後面那個眞正的內容，file4，卻是只有擁有者本人"可讀、可寫、不可執行"。照這樣看來，不是互相矛盾嗎？是有點兒奇怪，門牌的權限永遠是"rwxrwxrwx"，但那是假的，眞正的權限要看"門牌"後面那個檔案是什麼。上例的 file3 的權限，實際上應該是跟 file4 一樣："-rw-------"。

2.4 再接再厲

再繼續 *UNIX* 指令的學習：

chmod 是 change mode 的縮寫。用來改變檔案的權限，以前例 file1 而言，若要將它改成只有自己可以讀、寫，其他人（包括同 group 的人）一概不能讀、寫、執行，可以用：

```
% chmod 600 file1
```

這個三位數，600，每一位數字各代表擁有者（6）、同 group 的人（0）、其他人（0）的權限。

檔案擁有者的權限是"6"代表什麼意思呢？這個數字是用2進位算出來的：$6_{10} = 110_2$，用"1"表示權限被打開，"0"表示權限被關閉。所以110_2就表示："讀"的權限被打開、"寫"的權限被打開、"執行"的權限被關閉；也就是"rw-"。

同 group 、其他人的權限都是$0_{10} = 000_2$，表示讀、寫、執行的權限都被關閉（---）。同理，如果是"r--"就是$100_2 (= 4_{10})$（可讀、不可寫、不可執行）；"rwx"就是$111_2 (= 7_{10})$（可讀、可寫、可執行）。所以，你現在應該知道如何把一個檔案變成"rwxr-xr--"了吧？你也可以用同樣的方法去改變一個目錄的權限。另外，經過連結的檔案在改變權限時要注意一點，如前面的例子：

```
lrwxrwxrwx 1 john project1   6 Jul 22 09:22  file3 -> file4
-rw------- 1 john project1 212 Jul 22 14:23  file4
```

如果你對 file3 改變權限，事實上改變的會是 file4 ，你知道爲什麼嗎？

你可以隨便找個檔案和目錄來玩改變權限的遊戲，體會一下檔案權限對你使用上的影響。

你可以試著把一個目錄變成"d-wx------"，然後用"ls 該目錄名稱"來看目錄裡的東西；先用"cd 該目錄名稱"跳進那個目錄；再用"cd ./該目錄名稱"跳進那個目錄；看看各會得到什麼結果。改變該目錄的權限，看看結果又是如何[13]。

改變檔案權限，除了用數字外，還可以用文字的表示法，例如，要將 file1 改成"同 group 的人也可以寫(writable)"，可以用：

```
% chmod g+w file1
```

[13] 等你玩夠了，你大概要先休息一下，免得這些看似毫無道理的系統訊息把你學習 unix 的好興致磨光了！

"g" 表示 group，"＋"表示"加上"，"w"表示"寫"的權限。"檔案擁有者"的代號是"u"（user）；"同 group 的人"代號是"g"（group）；"其他人"是"o"（others）；"所有使用者"的代號是"a"（all）。"打開"某個權限，用"＋"號；"關閉"某個權限，用"－"號；"設定"某個權限，用"＝"號。舉幾個例子：

```
% chmod o-w file1   (關閉其他人 寫 的權限)
% chmod a-rw file2  (關閉所有使用者 讀、寫 的權限)
% chmod g+xw dir1   (打開同 group 的人 寫、執行 的權限)
% chmod u=rx file3  (設定檔案擁有者本人對 file3 可讀、可執行、不可寫)
% chmod u=rw,go=r file4  (跟 chmod 644 file4 完全一樣意思)
```

特別注意，使用"＝"號時，只影響被設定的對象，如上例的"u=rx"，只對"檔案擁有者" 的權限做改變："rx"表示"可讀、可執行"，沒有"w"，表示"不可寫"；非設定對象的 group、others，則不受影響，原來是什麼權限，還是不變。最後一個例子說明你可以同時對兩個以上的對象做權限的變更，注意例子中的"go" 是"group"和"others"合在一起寫的。

chmod 還有一個很重要的 option："-R"（注意，是大寫的"R"）。跟前面在說明"rm -R"指令同樣道理，是將一個目錄底下的所有檔案、子目錄，一層一層變更權限的意思，如：

```
% chmod -R ug+r,o-r,a+x dir1
```

你可以試著說出它的意思嗎？

which 用來找某個指令的所在位置（記不記得一開始提過， *UNIX* 裡頭的"指令"其實絕大多數是分散的程式）。例如，要找 "cat" 這個指令所在的位置，你可以用下列的用法：

```
% which cat
/bin/cat
```

系統會告訴你 cat 所在的路徑，把 /bin/cat 列出來。

特別強調，如果你是要找某個檔案在那個地方，"which"這個指令是沒這個能耐的。例如，你在以前編輯了一個檔案，叫做"my_letter"，現在忘了擺在哪個目錄下，想找出來，但是整個檔案系統有如茫茫大海，不知從何找起，總不能一個目錄一個目錄地"cd"進去，"ls"看看"my_letter"在不在那邊，不在的話，再跑到別的目錄找。*UNIX* 有聰明的方法來幫你做這種事，後面會提到"find"指令，就是讓你用來找東西的。你用"which my_letter"來找，是不行的；因爲它只認"可執行"檔，而且只在幾個特定的地方找，它的限制是：

which 完全依賴你的".cshrc"檔，而且只能用來找權限（permission）是"可執行"的檔案，找的順序依序是：

1 你要"which"的東西，有沒有在你的".cshrc"裡面被定義別名（alias）？

2 依照你在 ".cshrc"裡面所定的 path 順序去找，先找到的那個就是[14]。

比如說前面的例子，用了那麼久的"cat"，你想知道"cat"這個指令到底位在整個檔案系統的哪裡，所以你用"which cat"來找。系統在回答你之前，是做過一番努力的：

[14] path 和別名（alias）在 C Shell 那一章會詳細說明，如果你心急，想先知道它們是幹什麼的，可以先跳到那裡看一下，不過要記得回來這裡，把"which"這個指令看完；或是，你也可以硬著頭皮看下去，不清楚的都先不管，在讀完 C Shell 那一章後，再回頭看這一段

首先，它先看看在你的".cshrc"裡面有沒有把"cat"這個字眼定義成別名，有的話，它會直接回答你說"cat: aliased to 什麼什麼的"，意思是說，"cat"實際上是個別名，當你下指令"cat XYZ"時，系統實際上是執行"什麼什麼的 XYZ"；因爲在你的".cshrc"裡面定義了一個叫"cat"的別名。

如果在你的".cshrc"裡面沒有定義一個叫"cat"的別名，那麼系統就把你在".cshrc"裡面定義的"path"路徑，從第一個開始，一個一個跑進去找，看看有沒有一個叫"cat"的檔案，而且它的檔案權限是"可執行"。找到的話，就像上面的例子一樣，報告出來是在那個路徑下發現有個檔案權限是"可執行"的"cat"檔案；萬一，跑遍了所有的路徑都沒能找到叫"cat"的檔案，或者有找到叫"cat"的檔案，但它的檔案權限不是"可執行"，那麼，系統會回答你："no cat in xxxx yyy zzz ..."，其中的"xxxx yyy zzz "是你在".cshrc"裡面定義的 path。那表示系統沒有找到任何"cat"。

which 常用來讓你確認所要執行的指令正確無誤。例如，你下"cat .cshrc"指令，你以爲系統會把".cshrc"的內容顯示出來，但結果並不是這樣，或者，感覺結果怪怪的，跟預期結果不太一樣。你可以用"which cat"來確認你用的"cat"指令眞的是系統原來的"cat"（/bin/cat），而不是別名叫"cat"的指令，或在其它路徑下也叫做"cat"的程式。

你可以試試看，在你的".cshrc"裡面加上這一行：

```
alias cat ls
```

存好檔案，然後用"which cat"看看結果跟先前還沒加這一行時有何不同。你再把這一行改成：

```
alias cat ls -la
```

存好檔案，然後用"which cat"看看結果又有何不同。接著，把這一行從你的".cshrc"中拿掉，存好檔案，直接下指令：

```
% alias cat ls     或：
% alias cat ls -la
```

然後用"which cat"看看結果有何不同。再對照前面說的有關"which"的限制，你可以說出爲什麼這些看似相等的動作，結果卻不同嗎？

vi　visual editor 的縮寫。這是個令人又愛又恨的編輯器（Editor），我想沒有一個 *UNIX* 初學者會喜歡它（Well，如果你是一個 *UNIX* 初學者，而你馬上就喜歡用"vi"，那你眞的跟別人不一樣，非常不一樣！）。"vi"是 *UNIX* 裡面既定的（default）編輯器，功能很強，但是它的使用方式足以令一個初學者落荒而逃，所以在這裡先不做進一步介紹。意思當然不是要各位不必編輯檔案了，而是有其他功能也很強大，使用上更方便的編輯器可以用，不必再忍受 vi[15]。

ps　process status 的縮寫。用來看目前系統中正在跑的程式有哪些。就像前面說過的，*UNIX* 是多人多工的系統，當你在使用你的 *UNIX* 系統的同時，別人也可能在裡面跑程式， *UNIX* 提供這個指令"ps"，讓你看看系統目前的使用的情形。在 *UNIX* 裡面跑的程式，我們有時候稱它們爲"程序"（process）。

"ps"指令常用的選項有 4 個，是：a、u、x、w。

```
% ps -auxww
```

[15] 對一位初學者而言，或者應該說：會用的繼續用，不會用的不必學，除非你可能成爲系統管理人。

"a"(all)表示把別人的程序也列出來；"u"(user)表示要列出各程序的執行者；"x" 表示把沒有終端機直接控制的程序也列出來；"w"(wide)這個選項則可以把指令行很長的程序顯示出來，如果不加這個選項，因螢幕寬度有限，系統會自動切掉超過螢幕長度的部分，有時候就看不出眞正在背後跑的程式是什麼，"w"給得愈多，顯示的長度就愈長[16]。這個指令顯示的訊息，大概像這樣：

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
root         0  0.0  0.0    0    0 ?  D    May 10  1:03 swapper
jacky     7559  0.0  0.0   36    0 co IW   May 23  0:00 /bin/sh ..
john      1345 35.6  4.5   52    0 ?  R    May 24 12:45 /usr/local/
```

如果不給"x"選項，那麼第一個由"root"所擁有的程序就不會列出來；如果不給"a"選項，則只列出自己的程序；如果不給"w"選項，就像上面第二個由"jacky"這個使用者所擁有的程序 7559 就看不出到底是在跑那個程式，因爲整個指令行太長了，被截去後半段。如果不給"u"選項，那麼就不會顯示出最左邊一欄的使用者。這些輸出欄位，以第二項（process id，PID），第三項（該程序所佔用的 CPU 比例），第四項（該程序所佔用的系統記憶體比例）最重要，其它項目如果你有興趣知道，"man ps"吧！

一般，你最常用"ps"來看某個當掉的程式的"程序編號"（PID），再用下面要說的"kill"指令殺掉該程序。有時候你也會用"ps"來看是誰跟你一起搶機器用，由"%CPU" 那一欄看出是誰佔用 CPU 最多時間。"ps"也是少數幾個可以把選項的前導符號"-"省略的指令之一。

這裡要特別介紹一下 *UNIX* 裡所謂的程序（process）。基本上，程式跑起來後就是一個程序。而每一個程序，系統都會給予編號。程式跑完了，該程序就結束，也就是說，（用"ps"指令也）看不到了。*UNIX* 裡頭，有很多"跑不完"的程序，一開機就在跑，一直到關機爲止。這些程序維繫整個 *UNIX* 的運作，像監聽網路啦，等使用者 login 的啦，收、發 mail 的啦，等等。這些程序是永遠不停的，當然，大部分時間是在閒磕牙發呆（idle）。他們多數有個很奇怪的名字叫"魔鬼"（daemon）。這個名字的由來有很多傳說。聽說是沿用自 MULTICS[17]，也有人說是"Die And Execute MONitor"的縮寫。不過，眞正的由來，大概又得留給 *UNIX* 迷去考古了。

還有，在某些版本的 *UNIX* ，"ps"的選項差異很大，在 System V 系列的 *UNIX* ，你要用"ef"來代替上面說的"aux"。

kill 這個指令用來將某個正在跑的程序（process）殺掉。比如你跑一個程式，忽然發現給錯輸入資料，要重新再跑一遍；或者，跑著跑著，程式因某種原因當掉了；這時你就可以用"ps"指令找到該程式的程序編號(PID)，然後殺掉它。以前面的例子來說，你是"jacky"這個使用者，你想停掉編號"7559"那個程序：

```
% kill 7559
```

這裡的"7559"是你要殺掉的 process，千萬要給對，否則可能會誤殺好的程序。當然啦，自己的程序才能殺，別人的你管不到。有時候，你只用"kill 程序編號"還沒辦法殺掉該程序，這牽涉到程式本身接不接受"自殺令"的問題，可以試試"kill -9 程序編號"，強迫殺掉它。若這樣還不行，那就沒辦法了，找你的系統管理人來吧！

grep 這個指令功能強大，非常有用，一般的使用者因爲對所謂"regular expression"不熟悉，所以這裡只先介紹它最簡單的功能[18]：搜尋某一特定字串。 舉兩個簡單的例子：

[16] 這是對 SunOS 而言，其它 unix 系統可能有別的選項，例如，HP-UX 就要用"ps -ef"。
[17] 關於 MULTICS ，請看第三章。
[18] 詳細用法在第六章會介紹。

(1) 在檔案中，找出某個關鍵字：

```
% grep "printf" file1
% grep "My" file1 file2 file3
% grep -i "My" file1 file2 file3
% grep -n "My" file1 file2 file3
% grep -v "My" file1 file2 file3
```

第一個指令是說：在目前這個目錄下的檔案 file1 裡，找到含有"printf"這個字樣的那些行，並把它們（整行）印出來。

第二個指令是說：在目前這個目錄下的檔案 file1， file2， file3 裡，找到含有"My"這個字樣的那些行，並把它們（整行）印出來；印出來時，會標明哪些行是在哪個檔案。第三個指令和第二個指令一樣，但是忽略大、小寫，亦即"My"、"MY"、"mY"、"my"都會被找出來。

第四個指令和第二個指令一樣，但是，會把行號列出來。

第五個指令和第二個指令相反，它是列出"不含 My"這個字樣的那些行。

你可以試著將"i""n""v"組合來用 grep ，例如：

```
% grep -in "My" file1
```

(2) 在眾多 process 中，找出特定的某個。

```
% ps -auxww | grep textedit
```

這個指令是說：目前在系統中所有的 process 裡，找出含有"textedit"這個字串的那個 process。"|" 在 *UNIX* 中是另一個重要的觀念，稱爲管路（pipe）。在第 4 章會有詳細的介紹。

df disk free 的縮寫。這個指令用來顯示跟你用的機器有關的檔案系統的使用情形。以下例而言，它顯示了你所使用的系統，其檔案系統包含一個自身擁有的硬碟（/dev/sd0），分成三個檔案目錄(/，/usr，/spare)。另有兩個檔案目錄(/home/tool，/home/cadence) 則是由別的機器的檔案目錄"掛"(mount)過來的。各個檔案系統的容量及使用情況也都列出來了：

```
% df
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/sd0a              28703    4295   21538    17%    /
/dev/sd0g             215195  162776   30900    84%    /usr
/dev/sd0h             499710  337437  112302    75%    /spare
roxy:/export/tools
                      950755  595358  260321    70%    /home/tools
dylan:/export/cadence
                     1363950 1081046  146509    88%    /home/cadence
```

其中，第一欄顯示該檔案系統實際的位置，例如第一行的"/dev/sd0a"表示這個檔案系統是機器本身所擁有的編號爲"0"的磁碟的"a"區塊（註：一個磁碟有時可以再分爲（partition）數個區塊）。第四行的"roxy:/export/tools"表示這個檔案系統不是機器本身所有，而是"roxy"這部機器的磁碟，利用"NFS" 讓你的機器也"看"得到；這個檔案系統在"roxy"這部機器上叫"/export/tools"，但在你所用的這部機器上卻叫"/home/tools"，兩個名字不同，但實際上是同一個東西。當你存取"/home/tools"裡面的東西時，系統在背地裡是跑到"roxy"這部機器

的"/export/tools"去要東西的，只是這些動作你都感覺不到罷了。這是"NFS"的好處之一。

第二欄顯示該檔案系統總共有多少"kbytes"的磁碟空間。第三欄顯示該檔案系統已經被使用掉（used）多少。第四欄顯示該檔案系統還剩多少容量可用。第五欄顯示該檔案系統的使用率。第六欄顯示該檔案系統是"掛"在你所使用的這個系統的哪個路徑下。

眼尖的人可能注意到了，第三欄和第四欄的總和（"used"＋"avail"），並不等於第二欄的磁碟空間。這是因爲，磁碟的一部分（一般約 10%）被保留起來，作爲其它用途，只有系統管理人可以使用，一般的使用者無法使用到這些被保留的部分。所以，下次當你意外發現某個檔案系統的"capacity"一項居然超過 100% 時，就不必太驚訝了。而如果你發現系統向你發出"File system full"的訊息時，你應該知道要用哪個指令來查看是哪個磁碟"爆"了。

注意，System V 系列的 *UNIX* ，其"df"指令的輸出格式稍有不同，一般會有另一個叫"bdf"的指令（"b"表示"BSD"或"Berkeley"），其輸出格式就是上面所說的那樣。

du　disk usage 的縮寫。用來計算磁碟的使用量，以下例而言，它顯示了"/usr/bin"底下的東西共佔了 7324 KBytes。

```
% du -s /usr/bin
7324    /usr/bin
```

如果你要將每個子目錄所佔的使用量也個別顯示出來，可以不加"-s"選項：

```
% du /usr/bin
1500    /usr/bin/sunview1
7324    /usr/bin
```

第一行輸出結果告訴你在"/usr/bin"底下還有個目錄"sunview1"，這個子目錄用了 1500 KBytes；最後一行輸出告訴你，"/usr/bin"這個目錄（含其以下的所有檔案、子目錄）的用量總和是 7324 KBytes，就是你有加"-s"選項時看到的結果。

注意，有些系統的輸出數字可能是以"block"（512-byte爲一個"block"）爲單位，你可以"man du"看看你的系統是以什麼單位在計算。

telnet　這個指令用來與其他系統連線，例如你要與 roxy 這個機器連線，你可以用它的 hostname， 或是用它的 IP address 也可以：

```
% telnet roxy
% telnet 123.4.5.6
```

每個 *UNIX* 機器都有個名字，叫做它的"hostname"（你可以用"hostname"指令來看它的名字）。我們通常用這個名字來稱呼某架機器。這個名字是你的系統管理人取的，也因爲如此，在不同的兩個地方的 *UNIX* 機器，可能就會有相同的名字，就像到處都有同名同姓的人一樣，如果這兩個系統老死不相往來，那就沒什麼關係，萬一要互相溝通，那就很麻煩了，"你家的王小明"，"我家的王小明"這樣只有兩個系統溝通倒還好，全世界要溝通就有困難了。所以後來大家說好要有個區域地址來辨別："東二街12號2樓的王小明"，"西二街12號2樓的王小明"等等，就不會搞錯了。當然你不能在"東二街12號2樓"住了兩個王小明，那不行，系統管理人必須注意這一點。

上面所謂的"區域"，"專家們"有個專有名詞來表示："domain"。靠著這樣的分類，整個網路才能有秩序，送電子郵件時也才不會亂掉。

除了"名字"以外，你還可以用"數字"來表示一部機器，像身份證號碼一樣，所以每部機器都不一樣。例如在 Internet 這個目前堪稱全世界最廣爲使用的網路集合而言，你可以用：140.113.4.6、140.96.200.1 這樣的數字組合，來指定某個定點的機器，而不必知道它們的名字（和"區域"）。

上面說的"數字"，"專家們"也有個專有名詞來表示："IP"，或"IP Address"。"IP"是"Internet Protocol"的縮寫。

這是網路的基本概念，在這裡草草說了一點，不是很完整，如果你還想知道更多，可以再找相關書籍來看。上面說的應該夠讓初學的人得到一點概念了。當然，諸如"一個機器可以有兩個 IP"啦，等等容易招來"霧水"的話，還是留給別人去說，這裡就不提了。

rlogin　remote login 的縮寫。它與 telnet 的差別在於，telnet 永遠會問你 "username" 和"password"，但 rlogin 不問 username，如果你設定得宜，也可以不問 password（後面再說什麼叫"設定得宜"）。另外，rlogin 只能使用在遠端機器有你的帳號，而且該帳號與你目前用的系統相同的機器上。一般，rlogin 只使用在自己的實驗室、計算機中心，眞正遠端機台的 login 多是用 telnet。

rsh　remote shell 的縮寫。它又與 rlogin 不同。你必須有相同的 username 才能用"rsh 遠端機器"來進入該遠端系統。它的使用時機一般是要暫時到另一部機器去執行某個程式／指令。如：

```
% rsh roxy who
```

是到 roxy 這個機器去，看看有哪些使用者正在用該機器。要區別的是，在這裡，你並沒有眞正 login 到 roxy 去，只是去執行 who 指令而已。但，如果不給指令，只是打"rsh roxy"，那跟"rlogin roxy"是一樣的意思了。

find　這個指令同樣是一個功能強大，但一般使用者不太會用的指令[19]，這裡先教你下面這個用法就可以了："find dir -name filename -print"（"find 開始搜尋的目錄起點 -name 要找的檔案 -print"）。這是用來大海撈針的指令，它是說：從"dir"這個目錄開始往下一層一層去尋找 filename 這個檔案，然後把它的所在路徑印出來。當你的檔案系統越來越大、越複雜（子目錄一大堆），而你想找出 roxy.dat 這個檔案到底身在何處時，你非得用它不可：

```
% find ~ -name roxy.dat -print
/export/servTX/fake/users/jacky/roxy.dat
/export/servTX/fake/users/jacky/tmp/roxy.dat
```

compress　這是 *UNIX* 的標準壓縮程式，用來將一個檔案壓縮，壓縮後的檔案以原檔名加上".Z"（大寫的 Z）爲名。例如：

```
% compress roxy.dat    （會產生 roxy.dat.Z，是一個壓縮檔）
```

uncompress 這是 *UNIX* 的標準解壓縮程式，是上個指令的反動作，用來將一個檔案解壓縮，例如：

```
% uncompress roxy.dat.Z     （產生 roxy.dat）
```

要注意的是，壓縮過的檔要以".Z"爲檔案名字的結尾，否則"uncompress"不做事，因爲它不曉得要把解壓縮之後的檔取什麼名字。

[19] 所以，如果你很難記住它的使用法，也不必太沮喪。

uuencode *UNIX* to *UNIX* encode 的縮寫。它將一個 binary 檔案[20] 重新編碼成普通的 ASCII 檔案，以利傳輸。 使用時必須給一個標頭（label），下面是一個例子：

```
% uuencode roxy.exe abc > roxy.uu
```

上例將 roxy.exe 這個檔案編碼成 roxy.uu 這個檔案，標頭是"abc"，編碼完後，放到"roxy.uu"這個檔案。roxy.uu 的內容看起來像這樣：

```
begin 755 abc
M@0,!"P  X    "    3Y&         " @           "\$"  T .@0)(#H$25
M*B "E *@!)0"0 H7  ! U"+BN ,   C" &((@)   0*   0!    0   "@$
M  !  #@$ 0   $   Q2<(Z @0  X P$   !  #@$ 0   )WC0V@0   (KA7B
M*!$   B0$B(,0   2Y(0( "L$  (D@.@7$   $N4$" @@*(@(!*  $\!
...
...
M
0

end
```

你可以看到，第一行有三個東西，第一個字"begin"是表示這是整個 uuencode 檔案的開始；第二個數字是表示原始（未經編碼）檔案的權限是"755"；第三個字"abc"就是在指令中給的"label"，它的作用是告訴解碼程式，這個編碼過的檔案解出來後要叫什麼名字。通常，我們會給跟原來檔案一樣的名字："uuencode 檔案名字 檔案名字"，一來不必記"uuencode"後面要先給檔名，還是"label"；二來，在解壓縮時能還原成原來的名字。"uuencode"後的檔案最後會有一個"end"字樣，表示檔案到此為止。

因為 *UNIX* 原始的電子郵件系統無法正確傳送 binary 檔案，比如，已經編譯好（compiled）的程式，或中文信件等等。在送這些檔案之前，你必須用"uuencode"把它們先編碼，然後再 Email 。

"uuencode"的輸出原來是直接出現在螢幕上（記得 Standard Output 嗎？），所以我們用"＞"來將輸出結果"重導"（redirect）到另一個檔案去。

uudecode *UNIX* to *UNIX* decode 的縮寫。它將一個經 uuencode 編碼過的 ASCII 檔案，還原成原來的樣子。

```
% uudecode  roxy.uu
```

延續前面的例子，上面這個指令是將 roxy.uu 這個檔案解碼還原成 abc（標頭給的名字）。abc 與原來的 roxy.exe 是完全一樣的。

就像前面所說，uuencode 和 uudecode 的使用在利用 Email 傳送資料時非常有用，利用這兩個指令的配合，發送端先把檔案"uuencode"後再 Email，接收端再用"uudecode"將收到的檔案還原，解決 Email 無法正確傳送 binary 檔的問題。

w 這是 *UNIX* 系統中最短的指令，沒有其他指令像 w 這般簡單，只用一個字母。它告訴你目前系統的使用情形。以下是一個例子：

```
% w
  7:23pm  up 19 days,  9:28,  10 users,  load average: 0.05, 0.04, 0.05
User     tty       login@  idle   JCPU   PCPU  what
```

[20] 當然，如果你對一個普通的 ASCII 檔案做 uuencode 也是可以，多此一舉罷了。

```
John     ttyp0     4:04pm  2:22                -bin/csh
John     ttyp3     4:04pm  2:47      2      2  xterm
John     ttyq1     4:36pm  2:47      7      7  /usr/openwin/bin/xview/textedit
Jacky    ttyqa     6:24pm    17   4:24         vi BM_DOP_e.twr
```

這裡的輸出結果，第一行由左到右依序是：現在時刻；系統已經開機多久；目前有多少人 login 到這個機器；機器的平均負載。最後一項有三個數字，大體上，數字愈大表示機器愈忙碌，負載也愈大；反之，數字愈小表示機器愈閒（沒事做，負載愈小）。第二行以後，分別顯示各使用者的 username；從哪個終端機做事； login 的時間；已經多久沒碰鍵盤了；接著兩個數字是 CPU 所用時間，沒啥用處，不管；最後是該使用者正在做的事。

用"w"並不能看到所有的程序，這是要特別小心的[21]。

su　switch user 的縮寫。當你要借別人正在使用的機器 login，做些簡單的動作時，這個指令是你一定要用的。

```
% su - 你的username
Password: (輸入你的密碼)
```

當你正確輸入完你的密碼後，你所借用的那個殼（原來是別的使用者在用），就變成是你的了。還記得那個"who am i"指令嗎？ 每次使用"su"後，別忘了確認一下你是不是眞的進到系統了！

wc　word counting 的縮寫。它計算檔案的大小：幾個字元（character, -c）、幾個字（word, -w）、幾行（line, -l）。

```
% wc -wcl file1
967 7842 223 file1
```

"wc"只有三個選項，可以給一到三個，不給也可以；依照你在指令行所給的順序輸出結果，例如上面的"-wcl"表示，輸出的數字依序代表字數、字元數、行數。當你沒給任何選項時，輸出結果順序就是"lwc"。如果你只需要其中一個數字，比如，只要看字數，你可以用："wc -w file1"。

env　用來顯示各"環境變數"（Environment Variable）的設定值[22]。

diff　difference （differential file comparator）的縮寫。用來比較兩個檔案的不同。這個指令的輸出格式是爲了讓"ed"這個程式可以吃進去，然後將兩個檔案變成一樣而設計的，對於不熟悉"ed"指令的你來說，它的格式是有點讓人迷糊的，這裡的講解希望能讓大家易於解讀"diff"的輸出。現在假設有兩個檔案分別是"name.old"及"name"：

```
% cat name.old
Neil Young
Bob Dylan
James Taylor
Lindsey Bukingham
Sarah Hickman
Jackson Browne
```

[21] 那你大概又要問，這個指令跟前面講的"ps"和"who"有何不同？你自己比較看看，不就知道了嗎？

[22] 注意，設定環境變數是用"setenv"指令。而"set"指令則同時可用來設定及顯示各 C Shell "預設變數"（predefined variable）的設定值。

```
% cat name
Neil Young & Crazy Horse
Bob Seger
Bob  Dylan
James Taylor
Lindsey Bukingham
Jackson Browne
Paul Simon
```

現在用"diff"來看這兩個檔案的差異：

```
% diff name.old name
1,2c1,3
< Neil Young
< Bob Dylan
---
> Neil Young & Crazy Horse
> Bob seger
> Bob  Dylan
5d5
< Sarah Hickman
6a7
> Paul Simon
```

"diff"的輸出中有三個字母要先認識，它們是："a"，表示"append"或"add"；"c"，表示"change"；"d"，表示"delete"。在這些字母的前面和後面會有數字來表示行號，字母前面的行號是對應到"diff"指令的第一個檔案，字母後面的行號是對應到"diff"指令的第二個檔案。還有，如果有"c"字母出現，你會看到"<"及">"符號爲首的幾行字，分別表示第一個檔案原來的內容，及第二個檔案的內容。

來看上面的例子。輸出的第一行是"1,2c1,3"，表示：第一個檔案（name.old）的第 1～2 行要改變（"c"）成"下列文字"，才會跟第二個檔案的第 1～3 行完全一樣。所謂的"下列文字"：以"<"開頭的是第一個檔案的內容；：以">"開頭的是第二個檔案的內容；就是說，把第一個檔案的第 1～2 行用第二個檔案的第1～3 行取代，這兩個檔案的第 1～3 行就會完全一樣了，改變前的內容（計兩行），如"<"所標示，改變後就變成如">"所標示的一樣。中間的"---"只是用來將"<"和">"兩部份分開，讓輸出不會太亂。

再看，"5d5"。只有一個數字表示"只有那一行"的意思。"d"表示"除去"。"5d5"就是說，第一個檔案（name.old）的第 5 行要去掉，才會跟第二個檔案（name）一樣，要被除去的行列在下面，以"<"標示。"d"後面的"5"表示，經過前面（包括"1,2c1,3"和"5d5"）的處理後，現在的位置已經到第 5 行了。

再看"6a7"。"a"表示要"增加"。"6a7"就是說，第一個檔案（name.old）的第 6 行之後要增加一行，增加的那一行以">"標示在下面。"a"後面的數字"7"表示，經過這以上的種種改變，已經處理到第 2 個檔案的第 7 行。

要注意"diff"也會把空格當作普通的字元，所以"I␣like␣*UNIX* "跟"I␣like␣␣*UNIX* "會被"diff"認爲不同。如果你要叫"diff"忽略這些空白（包括 space 和 tab），你可以加上"-w"這個選項："diff -w file1 file2"。如果只是要忽略每一行最後多餘的空白，則用"-b"選項。另外，"-i"選項可以叫"diff"忽略大小寫的差異。

"diff"只能比較兩個檔案的不同，如要比較三個檔案，要用另外的"diff3"指令，在

此不多說明[23]。

cmp compare 的縮寫。用來比較兩個檔案是否相同。與"diff"不同的是："diff"比較的是 ASCII 檔案，列出詳細的不同處，而"cmp"則沒有限制，只單純地將兩個檔案一個 byte，一個 byte 地比較，然後顯示：相同或不相同。另外一個有比較功能的指令是"comm"（"common"的縮寫），它會把兩個檔案相同的地方顯示出來。這兩個指令在使用頻率上較"diff"少，這裡就不花費篇幅來解釋，有興趣知道詳細用法的讀者自己可以"man comm"。

tar tape archive 的縮寫。用來將檔案"包裹"起來，成爲一個大包包。打包後的東西我們一般都用"xxx.tar"（以".tar"結尾）來當檔名，以便可以一眼認出是包裹。常用的選項是："c"（表示"create"）、"v"（verbose）、"t"（table of contents）、"x"（extract）、"f"（file）。

分別以下列三個例子來說明：

```
% tar cvf my-files.tar /home/users/john/my-dir
% tar tvf my-files.tar
% tar xvf my-files.tar
```

第一個例子，是將"/home/users/john/my-dir"這個目錄以下的東西一層一層的全部打包起來，放在"my-files.tar"這個檔案中。注意 tar 這個指令的選項不加"-"。這個例子是"產生"包裹放在一個檔案（my-files.tar）中，所以用"c"和"f"選項，要產生的檔案名字要跟在"f"後面。"v"只是用來顯示打包過程而已，如果你不喜歡它邊打包邊說話，就不要加"v"。

第二個例子，是看看"my-files.tar"這個包裹裡面有哪些東西，只是看看而已，並不是眞的打開，所以用"t"選項，"f"後面也是緊跟著檔案包裹。

第三個例子，是將 my-files.tar 這個包裹眞正打開，將裡面的東西全部拿出來。要特別注意的是，tar 這個指令在打開包裹時，是完全按照當初打包時所記錄的路徑位置來放包裹內的東西的，以第一個例子來說，打包時給的路徑是 "/home/users/john/my-dir"，那在第三個例子打開時，就是照著放在"/home/users/john/my-dir"這個目錄中，若是在打開時沒有該目錄，系統會幫你造一個。但是，如果當初打包時所記錄的路徑（像這裡的"/home/users/john/my-dir"）你沒有寫的權限，那就沒辦法打開這個檔案包裹。例如：

```
% tar cvf tmp.tar /usr/bin
```

是將 /usr/bin 下的東西打包，放在 tmp.tar 中。你在解開 tmp.tar 時會發現它會把包裹內的東西放到 /usr/bin 下，這是因爲在打包時所記錄的路徑是 /usr/bin 的緣故。除非你可以寫"/usr/bin"這個目錄，否則系統不會讓你打開它。

tar 也可以只把某個目錄下的某些東西打包，例如：

```
% tar cvf tmp.tar k1.ps CTA/file1
```

上述動作是把"k1.ps"（在目前的路徑下）和"CTA/file1"（在"CTA"這個子目錄下）打包成"tmp.tar"這個檔案包裹。

tar 當然也可以在某個包裹檔案裡面，只把某些東西找出來，例如：

```
% tar xvf tmp.tar k1.ps
```

[23] 那麼，有沒有比較四個檔案的指令呢？沒有。

是把前一個例子的"k1.ps"從"tmp.tar"這個檔案包裹裡拿出來。

"tar"最常用的選項就這些了，其它的功能你大概不會用得著，這裡不多介紹。最後要提醒大家，檔案包裹打開，裡面的東西拿出來之後，原來的包裹還會繼續存在，如果確定沒有用了，要記得將它除去，免得佔用太多硬碟空間。

head 這個指令是把一個檔案的頭幾行印出來。選項就是行數，例如：

```
% head -20 file1
```

是把 file1 的前 20 行顯示出來（在螢幕上）。如果不給選項，系統的 default 值是"10"。

tail 跟前一個指令相反，這個指令是把一個檔案的倒數幾行印出來，選項也是行數，例如：

```
% tail -20 make.log
```

是把 make.log 的倒數 20 行顯示出來。它的 default 值也是"10"。

另一個好用的選項是"-f"，用來監視一個持續變化的檔案，例如你將某個程式丟到 background 去跑，該程式的輸出也被重導到"make.log"這個檔案，因爲已經被重導，所以在螢幕上就看不到結果，你當然可以隨時用"cat make.log"或是"tail make.log"來看結果，但是你又不知道"make.log"是不是已經有新的資料進去，所以你也無法知道什麼時候該看結果了，這時候你可以用：

```
% tail -f make.log
```

來觀察"make.log"的變化。注意加不加"-f"用法的差異：不加，只是將指令動作"當時"的 make.log 的後 20 行印出來，便不再作用；而加上"-f"時，卻會一直印出 make.log 的後幾行，一直到該檔案不再變化爲止；這時候，你應該按下"Control-C"來中斷"tail -f"的執行，不然它會一直停在那裡。

cut 這個指令顧名思義，是用來"切"檔案的；而且它是"直著切"。切法就用選項來設定，以下面的幾個例子來說明：

```
% cut -c1-20 file1
% cut -c21- file1
% cut -f2,4 -d' ' file1 > file2
```

第一個例子，是把 file1 每一行的前 20 個字元切下來。"-c"的"c"表示 "character"，字元。第二個例子，是把 file1 每一行的第 21 個（含）以後的每個字元都切下來。"-c21-"，在 21 之後沒給終點數字，就是到那一行最後的意思，同理，第一個例子的"1-20"也可以省略成"-20"。第三個例子，是把 file1 每一行的第 2 、4 欄切下來，放到 file2 中。"-f"的"f"表示 field，欄位，而欄位的分界（"-d"的"d"表示"delimiter"）定義是以' '（空格）來分的，也就是說，碰到空格就算是另一個欄位了。這個指令在整理資料時非常有用，詳細用法和例子，在後面第六章會討論。

paste 前面說過 cat 指令可以把檔案連接起來，頭尾相接。而 paste 則是將兩個檔案"平行"地接起來。第一行對第一行，第二行對第二行...在比較過前一個"cut"指令後，你會發現，"paste"和"cut"的作用有點互補的味道。它的基本用法和例子我們在第六章會討論。

```
% paste file1 file2
% paste -d':' file1 file2
```

同樣的，"cat" 也有互補的指令：

split 既然有 cat 指令 可以把檔案上下連接起來，當然也有指令可以將一個檔案再拆開。split 是把檔案橫著切成一塊一塊，選項就是行數。如：

```
% split -500 file1
```

是把 file1 以 500 行爲單位，切成一塊一塊。切成的小塊檔案的名字是系統給的，以"aa"，"ab"，"ac"...爲結尾。另外還有一個叫"csplit"的指令，也是用來切割檔案的。

tr translate 的縮寫。這個用來將檔案中的字元按照下指令時給的轉換方法來做改變。例如，下面的例子是將"file1"中的小寫字母，全部改成大寫，然後再放到"file2"中：

```
% tr 'a-z' 'A-Z' file1 > file2
```

這個指令在第六章會詳細介紹。

sort 顧名思義，這個指令將檔案中的資料按照下指令時給的規則排序。例如，將檔案的內容按字母（character）順序排列：

```
% sort -c file1
```

這個指令還有很多選項，本書不多做介紹，老話一句，請有需要的讀者自己"man sort"。

cc C Compiler 的縮寫。是系統內建的 C 語言編譯器。每個 *UNIX* 系統中，C Compiler 的選項都不太一樣，你應該自己"man cc"以確定你該使用哪些 option。

whereis 用來找尋某個指令、library、和 manpage 的所在地方。例如：

```
% whereis cat
cat: /usr/5bin/cat /usr/bin/cat /usr/man/man1/cat.1v
```

請特別注意它跟"which"和"find"是不同的，不要混淆了。事實上，這個指令在使用上並沒有"which"和"find"來的多。

nslookup 用來查詢機器的註册資料。最常用來查詢某部機器相對應的 IP address。一個例子：

```
% nslookup ccca.nctu.edu.tw
Server:  oax2.ccl.itri.org.tw
Address:  140.96.110.211

Name:    ccca.nctu.edu.tw
Address:  140.113.5.150
```

你可以在你的機器上試試上面這個指令。

tee T型管。前面說過，可以用">"來把原先輸出到螢幕（標準輸出裝置）的東西，輸出到一個檔案，我們將它叫做"輸出重導向"。而經此重導向的結果就不會在螢幕上再顯示出來了。如果你希望那些結果在存放到檔案的同時也輸出到螢幕上，則可以利用"tee"這個指令：

```
% cat file1 file2 | tee file3
```

這個例子是說：把 file1 file2 連接起來，顯示在螢幕上，而同時也把這些結果放到 file3 這個檔案。因爲它的作用像是一個T型管，所以就取"T"的發音成爲指令名稱。

file 這個指令用來顯示檔案的型態。試著用下面這個指令就知道究竟了：

```
% file /usr/bin/*
% file /etc/*
```

2.5 魔數

來談談 *UNIX* 怎麼判別一個檔案的型態。

之前，我們說過可以用"chmod"指令來改變一個檔案的屬性。你可以將一個原來是"可執行檔"（executable）變成只能讀、不能執行；你也可以把一個普通文字檔的屬性，變成"可執行"。問題來了：當你把一個普通文字檔，改變其屬性成"可執行"後，你再把它眞當成"可執行檔"，或一個指令去執行它時，*UNIX* 如何去跑這個程式？（這個"可執行檔"明明是普通的文字檔，你硬把它改變屬性的）

UNIX 最聽話了，只要主人說是"可執行"，它就眞的當做"可執行"，然後用力去執行這個（不管跑得動或跑不動的）檔案。

怎麼"用力"法呢？首先，*UNIX* 先看看這個檔案的最前面幾個 Bytes 是什麼，看看這幾個 Bytes 長的樣子它認不認得。怎麼認呢？*UNIX* 記性也不好，它把它認識的東西都記在一個叫"magic"的檔案裡面（/etc/magic），這個檔案完整記錄了 *UNIX* 所認得的檔案型態，當你用"file"指令來檢視檔案型態時，事實上 *UNIX* 也是偷看這個檔案，然後告訴你答案。如果你要它辨認的檔案很奇怪，在"/etc/magic"中沒有記錄，那 *UNIX* 也是沒轍，胡亂猜一通，或乾脆說那個檔案型態是"data"。所以，現在你知道"file"這個指令和"/etc/magic"的關係了吧。我們再來看看"/etc/magic"這個檔案的內容，底下是截取一小部分 SunOS 的"/etc/magic"：

```
0 long 0101555 PDP-11 single precision APL workspace
0 long 0101554 PDP-11 double precision APL workspace
0 string \037\235 compressed data
>2 byte&0x1f x %d bits
0 short 0433 Curses screen image
0 string <ar>System V Release 1 archive
0 long 0x1010101 MMDF mailbox
0 string <!OPS Interleaf ASCII document
0 string %! PostScript document
0 string <MakerFile Frame Maker document
0 string From mail folder
```

從這個檔案可以推斷，如果一個檔案的最前面兩個字元是"%!"的話，*UNIX* 會當它是一個所謂的"PostScript"文件；如果最前面四個字元是"From"的話，*UNIX* 會把它當成是一個電子郵件信箱（mail folder）。如果你用的 *UNIX* 是 SunOS 的話，你可以做個實驗試試看：編輯一個檔案，在最開頭只寫四個字母："From me to you"，然後用"file"指令看看 *UNIX* 如何認這個檔案；把"From"改成"␣From"（在 F 之前多一個空格），再看 *UNIX* 認不認得；如果檔案的第一行是空白，第二行才是"From"時，又如何呢？

如果你不是使用 SunOS，試試用"%!"來當檔案開頭，加些空格、空行，看看有什麼發現。玩夠了以後，你大概也知道"file"指令，在某方面來說是一點也不可靠的。像下面這個檔案：

```
TARGET = p2s2
CFLAGS = -O
SOURCES = main_p2s2.c p2s2.c
SETLIB = -lm
OFILES = $(SOURCES:%.c=%.o)
p2s2: $(OFILES)
        cc $(CFLAGS) $(OFILES) $(SETLIB) -o $(TARGET)
.c.o:
        cc -c $<
main_p2s2.o : main_p2s2.c
p2s2.0 : p2s2.c
```

眼力好一點的 C 程式設計者大概可以看出它是一個 Makefile，可是 *UNIX* 硬說它是什麼"[nt]roff, tbl, or eqn input text"；如果你是用 SunOS，讓你大開眼界的事情就更多了。

除了在"/etc/magic"定義的以外，有一種檔案開頭的樣式是 *UNIX* 特別注意的："#!"。

當檔案的前兩個字元是"#!"時，*UNIX* 會認爲，緊接在"#!"之後的是眞正要跑的程式，而在第一行以後的文字，都要由緊接在"#!"之後的那個程式來解釋和執行。舉個例子：

```
% cat ./run
#!/bin/csh
echo I Love UNIX
% ./run
I Love UNIX
```

比如說這個檔案叫做"run"，當你輸入"./run"的時候，系統看到"run"的第一行的前兩個字元是"#!"，它就知道這個檔案的其它內容都要由緊接在"#!"之後的程式（"/bin/csh"）來解釋、執行。當然，這個檔案的屬性得先要變成"可執行"才可以， 否則你會得到"permission denied" 的訊息。

而這個例子：

```
#!/bin/sh
echo I Love UNIX
```

也是同樣道理。

有幾點要特別提出來。第一，如果最前面兩個字元不是"#!"，而檔案屬性已經是"可執行"，那麼，當你執行它時，*UNIX* 是假設"#!/bin/sh"，也就是說，把檔案內容當成是 Bourne Shell 的指令來執行（記得嗎？ *UNIX* 的原始殼是/bin/sh，Bourne Shell）。

所以，下面五種檔案開頭（第一行）方式，對 *UNIX* 來說是一樣的：

```
#!/bin/sh
#
#/bin/csh
# I Love UNIX
(空白行)
```

它們都被當成是 Bourne Shell 的指令。

切記養成好習慣：在檔案的第一行明確地告訴 *UNIX* 該檔案的內容要用什麼東西解釋。

第二，很多用過 DOS 的使用者常問：在 *UNIX* 裡面怎麼寫批次檔（batch file）。現在你會了嗎？下面這個"批次檔"的動作你能一步一步解釋嗎？

```
#!/bin/csh
cd
cp .cshrc .cshrc.back
cat .cshrc .cshrc.back > CSHRC.double
set lines = `wc -l CSHRC.double`
# 用“#”來做注解
# 用“set”指令將變數“lines”設爲“wc -l CSHRC.double”指令的結果
echo I have $lines lines in this file : CSHRC.double
echo I use echo to print something on the screen.
echo ""
echo ""
```

這就是一個"批次檔"了，如果你寫過其它電腦語言的程式，應該可以很快瞭解這一小段程式的動作，如果還不清楚某些行的意思，猜猜看應該也猜得出來。在 *UNIX* 中，我們不叫它"批次檔"，而稱它爲"script"。第一行表明是"#!/bin/csh"，是用 C Shell 來解釋的，所以叫" C Shell Script"。在第七章會教大家如何寫" C Shell Script"。

2.6 快學完了

剩下沒幾個指令了，加油！

touch 這也是個有用的指令，被它碰到的檔案，其檔案時間就會更新成目前時間。如：

```
% touch file1 file2 file3
```

會將 file1 、 file2 、 file3 的時間更新。如果要"touch"的檔案還未存在，系統會造個那個名字的新檔，檔案大小是"0"。寫程式的人如果常用 makefile，會發現用"touch"來強迫更新檔案時間能省事很多。

ln 這是製造連結指令。有關"連結"，在前面已經敘述很多，至於"連結" 的產生方式，這裡先舉個例子:

```
% ln -s BACKUP backup
% ls -la BACKUP backup
-rw-r--r--  1 john          8124 Sep 22 12:06 BACKUP
lrwxrwxrwx  1 john             6 Sep 22 12:06 backup -> BACKUP
```

上面的例子中，選項"-s"表示是做"symbolic link"，告訴系統做一個"從 backup 到 BACKUP"的連結。注意指令的順序，前面的"BACKUP"這個檔案（或目錄）是原來就存在的，而後面的名字"backup"本來是沒有的，這裡爲了方便，讓自己也可以用"backup"這個名字來存取"BACKUP"的東西，就造個門牌叫"backup"，這個門牌只是另一個門面，內容跟"BACKUP"是完完全全一樣的。經過這個連結動作後，不管你用"BACKUP"這個名字或者是"backup"，都是指同一個檔案（或目錄）："BACKUP"。連結的好處很多，靠各位慢慢體會了。

要拿掉連結，不管是目錄或檔案，都用"rm"指令即可。有些系統有"unlink"指令，也是拿掉連結的意思。要注意的是拿掉連結並不影響原來的檔案或目錄:

```
% rm backup
% ls -la BACKUP backup
backup not found
-rw-r--r--  1 john          8124 Sep 22 12:06 BACKUP
```

因爲 backup 只是連結，"rm backup"只是將"門牌"（名叫"backup"）拆除而已，原來的房子(名叫"BACKUP") 還是在的，不受影響。

再說明一點，這裡說的連結，指的都是較常使用的"symbolic link"（用"ln -s"做的連結）。另外還有所謂"hard link"（用"ln"，無選項，做的連結），有興趣的人可以"man ln"。

expand 這個指令用來將對位字元（TAB）轉成適當個數的空白（space）。在編輯檔案時，我們常用對位字元來讓內容看起來較整齊、美觀，例如，下面這個例子：

```
Name◇◇          Phone_Number◇  Address
王小華◇◇         02-1234567◇    台北
李中堅◇◇         03-7654321◇    桃園
陳大鵬◇◇         07-1234567◇    高雄
```

上面的檔案中，◇代表一個 TAB 鍵，用來使欄位的分隔整齊。在列印或某些情況下，你想將這些對位字元都換成適當個數的空白。由於對位字元所對應的空白個數並不一定，用普通的"find and replace"方法會因爲要取代的空白個數不固定，而無法達到目的， *UNIX* 提供這個"expand"指令讓你達到。

這個指令是屬於 BSD 的加強功能，所以是放在 /usr/ucb 之下。

```
% expand input_file > output_file
```

它的輸出是內定爲標準輸出，所以將它的輸出重導到一個檔案。你用"cat"、"more"等指令列出來時，看不出這種代換的效果，但用"vi"時就很明顯了。

"expand"有個相反指令，叫做"unexpand"，做的事情恰好相反。

uname *UNIX* Name 的縮寫。這個指令告訴你，你現在用的這個 *UNIX* 系統的名字，常用的選項是"-a"：

```
% uname -a
SunOS roxy 4.1.1 1 sun4c
```

你現在可以試試看你的機器是哪個作業系統（OS，Operating System）。

expr 這個程式最常被用來做加減乘除的運算，例如：

```
% expr 1 + 2
3
%expr 4 - 5
-1
% expr 1 \* 5
5
% expr 34 / 7
4
```

注意第三個例子的"*"，是因爲要避開"*"在 C Shell 中的特殊意義（"*"表示所有檔案），所以要加上倒斜線。特別提醒你，"expr"只表示整數，如果你需要浮點數運算，要用"bc -l"這個指令：

```
% echo 34 / 7 | bc -l
4.85714285714285714285
```

注意"-l"選項一定要加，否則也是整數運算結果。

還有一些更高深的指令像"sed"，"awk"，都不是三言兩語可以說完，在這裡沒有介紹，一來怕嚇跑初學 *UNIX* 的讀者，二者，筆者能力有限，不敢亂教，就留待其他先進出來指導了！

這一章大致介紹了一些較基本、較常用的 *UNIX* 指令，內容有點多，但千萬不要被嚇到了，還是一句老話：不必全記住，看過、有個印象就可以了。經驗勝過一切，看不懂的地方，自己上機試試會更有概念。

用過 *UNIX* 一段時間之後，你會發現 *UNIX* 的指令都很短，錯誤訊息也都很簡要，有時更讓人不知所以。其實這也是有原因的。*UNIX* 在發展的時候，終端機技術還沒有現在那麼好，反應也很慢。爲了遷就這個限制，指令當然不能太長，免得使用者花太多時間在打字上面，更何況，字母愈多，打錯的機會就愈大。一般的 *UNIX* 使用者都不是打字高手，有的甚至練就一手"一指神功"，只用一根手指頭打字、下指令，所以，指令宜短不宜長（"move"出錯的機率就比"mv"大，你說是不是？）。

錯誤訊息呢？如果太多太長，使用者就會把時間耗在"看完"這些錯誤訊息上，當時終端機速度那麼慢，顯示完一行可能就要讓使用者枯坐幾十秒，如果只是因爲指令下錯，就要讓使用者看著 *UNIX* 長篇大論的把顯示器塗滿，那不是太沒效率了嗎？所以，錯誤訊息不能太多，夠用就好。

2.7 1752 年 9 月

UNIX 有個指令叫"cal"，是"calendar"的縮寫，可以列出月曆、年曆。例如：

```
% cal 7 1995
   July 1995
 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

上面的例子是列出 1995 年 7 月的月曆，如果你要其它年、月，在"cal"後面給它適當的數字就可以了，若是在"cal"後面沒有其它數字，它列出的是目前系統時間當月的月曆，若是在"cal"後面只有一個數字，它會把那個數字當成年份，印出該年的年曆。但是，當你用下面這個指令時，你一定會大吃一驚：

```
% cal 9 1752
   September 1752
```

```
 S  M Tu  W Th  F  S
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

全世界的 *UNIX* 都一樣：1752 年的 9 月，居然少了 11 天！千萬不要以爲 *UNIX* 瘋了，或是哪個程式設計師故意開玩笑，那一年，教皇修訂曆法，將凱撒曆 (Julian calendar) 改爲現在的陽曆[24]。所以，那是眞的，1752 年沒有完整的 9 月！

介紹完常用指令，下一章我們將簡單回顧 *UNIX* 的輝煌歷史，輕鬆一下。

I wanted to design something that would
still be usable in 100 years. In other words,
my goal was to arrange things so that,
if book specifications are saved now,
our descendants should be able to produce
an equivalent book in year 2086 ...

from TUGboat,7(1986), 95-98.
by Donald E. Knuth

[24] 史稱"The Gregorian Reformation"。

Chapter 3

UNIX 簡史

3.1 失敗的計劃

時間回到 1968 年， *UNIX* 誕生的前一年。 *UNIX* 之父 Ken Thompson 與同事正在貝爾實驗室參與一個名爲 MULTICS 的計劃。那也是一個與電腦有關的計劃，由貝爾實驗室和 GE 兩家公司合作。不過方向有些問題，所需的硬體太大，即使有其他迷人的優點，也不免要走向結束的命運。Ken Thompson 不願心血白流，於是向上級要求要一部 DEC-10 來重建一個像 MULTICS 一樣，有很好的操作環境的作業系統，管理階層當然否決了他的提議，誰會將一個失敗的計劃在不同的硬體設備再重複一次呢？而當時，他已經有一個稱爲星際旅行的程式在 GE 635 的機器上跑，但是反應非常的慢！正巧，他發現了那部置於牆角的 PDP-7 ，Ken Thompson 和 Dennis Richie 就把那個星際旅行的程式移植到 PDP-7 上，時間是 1969 年。而這部 PDP-7 就此注定要在整個電腦歷史上留名...。

MULTICS 其實是"MULTiplexed Information and Computing System"的縮寫[1] ，而在 1970 年那個時候，那部 PDP-7 卻只能支援 2 個使用者，所以 Brian Kernighan 就開玩笑的戲稱他們的系統其實是："UNiplexed Information and Computing System"，縮寫一"UNICS"，然後，大家取其諧音，就叫它"*UNIX* "了。這個名字就沿用到今天。

3.2 誕生

後來他們又向上級申請一部 PDP-11/20 ，這一次，他們學乖了，申請的名義是：要發展文書處理系統。有了具體的方向，這個提案獲得採納，他們也眞的發展了一套文書處理系統，就是現在 *UNIX* 裡面的文書處理系統（nroff/troff）的前身！有趣的是，沒多久貝爾實驗室的專利部門眞的採用了這套系統作爲他們處理文件的工具，而貝爾實驗室的專利部門也就順理成章地成爲 *UNIX* 的第一個正式使用者了；時間是 1971 年，這部 PDP-11/20 只有 0.5 MB 磁碟。而描述這整個系統的文件被標示爲："First Edition"，日期是 1970 年 11 月。從此以後， *UNIX* 的版本就以系統文件的版別來稱呼。

1973年，Ken Thompson 和 Dennis Richie 將 *UNIX* 整個用 C 改寫，此時是第 4 版了。而 C 的前身叫 B ，也是他們倆個人的發明，爲了改寫 *UNIX* ，他們將 B 語言做了一番改進，成爲 C 語言。這個劃時代的動作讓 *UNIX* 能夠很輕易的被移植到各種不同的機器上，只要修改少許原始程

[1] 雖然有人戲稱它是：Many Unnecessarily Large Tables In Core Simultaneously ... 。

式碼，整個 *UNIX* 可以在很短時間內完成移植的動作。1974年，他們在"Communications of the ACM"正式發表了 *UNIX* ，吸引了更多學術界的注意與參與。 1975 年 第六版（6th Edition）發表，這是眞正流通到貝爾實驗室外的版本。1977年，UC Berkeley 開始公開 Pascal 翻譯器，連帶的發布了其他對 6th Edition 的改進程式碼，這是所謂的"1 BSD"！（1st Berkeley Software Distribution），也開創了 *UNIX* 的另一個分支： BSD 系列。另一個重要的歷史因素是，在 *UNIX* 的誕生初期，貝爾實驗室還是一個大的集合體，包括貝爾電話公司以及西方電子（Western Electric Co.）。礙於反托拉斯法的規定，它不能利用 *UNIX* 做任何商業行爲，也因此，內部的管理階層自然沒有興趣對 *UNIX* 有任何的支持， *UNIX* 就全靠一群玩家在琢磨，主要人物當然還是 Dennis Richie 和 Ken Thompson。1979 年左右 Ken Thompson 在柏克萊教授作業系統的課程，也造就了一些學生成爲 *UNIX* 史上的重要人物。

3.3 欣欣向榮

後來，對 *UNIX* 的需求漸漸多了，貝爾實驗室便定了一個簡單的授權規則：要 *UNIX* 可以，但沒有任何支援、試用、保證、不准廣告、亦沒有任何錯誤修定。

再後來，連自己公司內部也有越來越多的人在使用 *UNIX* ，貝爾實驗室便成立了一個專門支援 *UNIX* 的團隊，叫 *UNIX* Support Group，USG 。但這個團隊只准做支援的工作，不能做發展 *UNIX* 的工作。這個規定導致了每個使用 *UNIX* 的地方都在發展自己的 *UNIX* ！更何況， *UNIX* 的原始程式又都隨著 *UNIX* 一起發布出去，*UNIX* 在後來爲什麼會有那麼多版本，這是一個重要的關鍵。

1979年，第 7 版（V7）正式發表，包含了 K&R C Compiler 及 Shell（也就是 Bourne Shell）。這個版本被稱爲最後眞正的 *UNIX* （last true Unix），Stephen Bourne，Bourne Shell 的原作者之一稱它是"an improvement over all preceding and following Unices"。

由 V7，Bell Lab 發展了一個稱爲 32V 的改版，是 *UNIX* 跑在 32-bit VAX 機器上的版本，名稱即由此而來。而在發展時爲了簡化，他們將這個 32 位元的電腦當成是 PDP-11 的延伸，忽略了記憶體定址的硬體限制，也就是說，允許程式定址到實際硬體上所沒有的記憶體位址 ，這個功能就是有名的虛擬記憶體（Virtual memory）。柏克萊很快將這個機制加入到他們的 *UNIX* 內，而成爲 3 BSD ，也因此， BSD 系列的 *UNIX* ，其核心（kernel）就取名叫"vmunix"（Virtual Memory *UNIX* ）。柏克萊的 *UNIX* 發展團隊叫 Computer System Research Group（CSRG），這個隊伍替 *UNIX* 的成長立下不少汗馬功勞，1980 年發表了 4 BSD ，1983 年的 4.2 BSD 加進了 TCP/IP，1986年的 4.3 BSD 改進了 TCP 的效能，1988 年有 4.3 BSD Tahoe，1990 年的 4.3 BSD Reno，1992 年發表 4.4 BSD 。在這一年，CSRG 正式關閉，所謂 BSD *UNIX* 也正式走進歷史。目前，學術性的 *UNIX* 發展大概以在卡內基．梅隆大學的 Mach 系統最有名，也最有成就。原始的 *UNIX* 裡面，系統的指令程式是放在 /bin 這個目錄之下的，而後，BSD 系列所增加的指令（程式）就放在 /usr/ucb 之下，以便有所區別。這就是 /usr/ucb 這個目錄的由來。

貝爾實驗室的另一個 *UNIX* 發展部門叫 Programmer's WorkBench(PWB)，也發展了另一系列的 *UNIX* （PWB/*UNIX* ），連同 Dennis Richie 和 Ken Thompson，加上外頭學術界的努力， *UNIX* 的版本就愈來愈多了。後來（1982年）USG 和 PWB 合併成一個團隊叫 *UNIX* System Development Lab.（USDL），第一個產出物又是另一版 *UNIX* ： *UNIX* System III。這個版本是最後一個透過西方電子公司授權的 *UNIX* 。而後在 1984 年，因爲一項美國政府的反托拉斯判決，AT&T 被拆解成幾個子公司，在這之前 *UNIX* 已被轉到 AT&T 的 Information Systems，並在 1983 年發表了 System V。在 1984 年的判決後，AT&T 終於可以大張旗鼓的爲 *UNIX* 商品化，並且認眞的做統一 *UNIX* 和支援 *UNIX* 的事情。1984年，AT&T 發表 *UNIX* System V Release 2（SVR2），1986 年的 SVR3 也相當成功，1988 年更結合了 4.2 BSD 及 SunOS 的加強功能成爲 SVR4。

1991 年 4 月， AT&T 成立了子公司 *UNIX* System Laboratories(USL)，*UNIX* 這個商標

便由 USL 所擁有。1993年，網路巨人 Novell 公司買下 USL，然後將 *UNIX* 商標讓渡給了 X/Open。*UNIX* 的未來正如他的過去一樣令人目不暇給，至於 *UNIX* 的統一大業，誰也不敢預言，畢竟，有了過去二十年的紛紛擾擾，也才有今天成熟好用的 *UNIX* 。

3.4 Richard M. Stallman

你一定沒有想到，這個世界上居然有人窮其一生在寫程式給人拷貝，完全免費！不管你有沒有聽過 GNU ，現在開始你將認識它...。

Richard M. Stallman 原來是麻省理工學院人工智慧實驗室的一員，整個七零年代都在那裡。當時他們熱衷於程式設計和電腦，利用一部叫 Lisp Machine 的系統來研究人工智慧，研究氣氛相當熱烈，當時只要任何時候有人發現程式的 bug，馬上會有人來 fix，因此程式的發展極快。後來（1980）有人離開了 MIT 組了一家叫 Lisp Machine Inc.（簡稱，LMI）公司，另一票人則開了 Symbolics 公司。兩家都是向 MIT 簽約使用 Lisp Machine 的作業系統，也都同意：任何在 Lisp Machine 的改進，都歸 AI 實驗室擁有。這樣可以確保整個研發的進步與資源共享，就像還在 AI 實驗室時一樣。兩年後，Symbolics 的律師竟然在合約上鑽出了漏洞，認爲合約上只是說 MIT 可以擁有兩家公司的研究成果，卻不賦予 MIT 將此成果再散布出去，因此他們可以禁止 MIT 將他們的成果與 LMI 分享！這樣的舉動已經違背了原來合約的目的，軟體資源再也不能夠共享。Lisp Machine 的發明人 Richard Greenblatt 回憶道，他和 Richard Stallman 實在是失望透了，於是決定跟 Symbolics 誓不兩立，兩人日以繼夜花了兩個禮拜的時間爲 Lisp Machine 寫了一個比 Symbolics 還好的新作業系統，在以後的日子裡，只要 Symbolics 有任何新的東西，Stallman 就替 MIT 和 LMI 寫個更好的，而他通常只花費少許時間就辦到了，他的目的就是要懲罰 Symbolics 的背信。然而就在此時，他熱愛的 AI 實驗室裡，研究人員受到市場的吸引也一個一個走了，最後 Lisp Machine 也沒人管了，任其荒廢，Richard Stallman 終於意識到他眞正的敵人其實不是 Symbolics，而是整個軟體業界！他常舉的例子是 AI 實驗室的印表機。

Xerox 公司曾送了一部印表機給 AI 實驗室，當時的印表機非常龐大，因此研究人員把它放在 9 樓，和平常工作地點並不在一起，而靠網路連接。唯一的困擾是，他們得跑上跑下去設定印表程序，而當機器夾紙，或者是報表用完了的時候，他們也不會知道。當時，這部印表機附了原始程式碼（source code），所以他們就自己修改，讓它可以更聰明一點，總算解決問題。

到了1978 年 Xerox 又來了一部印表機，但不再附 source code 了。Richard Stallman 回憶說，那時沒辦法改程式，所以印表效率又變糟了。原因就在於：沒有 source code！Richard Stallman 舉例說，就像你買了一棟房子，當你的房子出了毛病時，限制你非得要找到原來的建造者來修不可一樣，你得一輩子被限制得死死的，被他牽著鼻子走。這不是很荒謬嗎？而 Richard Stallman 無法忍受這樣的事！他決心要打破這個軟體神話，將使用電腦軟體的人從軟體公司的手中解放出來！於是，在 1984 年，他離開了 MIT，立志要再造一個新的作業系統，讓所有的使用者不必再受限於某個特定的電腦硬體或者作業系統，由於 *UNIX* 的可攜性及普及，而且， *UNIX* 是由大大小小的獨立程式所組成，他選定 *UNIX* 作爲他的目標，而他把整個計劃叫做"GNU's Not *UNIX* "，簡稱"GNU"。

前面說過，柏克萊大學的 CSRG 也從事 *UNIX* 的改進工作，並且也散布 *UNIX* ，含 source code。但是由於 *UNIX* 是由 AT&T 所擁有，所以他們還得花 400 美金向貝爾實驗室買 *UNIX* 然後再修改它，也因此，即使你是向柏克萊買 *UNIX* ，你還是得再向貝爾實驗室買 *UNIX* 的授權，後來 CSRG 也因經費和授權的問題而被迫關閉了。但是 CSRG 的貢獻是有目共睹的，由於程式碼可以取得，每個人都可以修改，都可以發現缺點，於是問題就很快得到解決。程式碼的公開其實是軟體進步的先決條件，Richard Stallman 正是要這樣做，它要把 *UNIX* 裡頭的每個程式都替換掉，然後再把核心(kernel)也取代掉，把程式碼也都公開來，這樣一來，大家就可以有完全免費的 *UNIX* 可以用，而這個 *UNIX* 不是老的 *UNIX* ，是完全不必付錢的 *UNIX* 、不必再受制於 AT&T 的 *UNIX* ！

Richard Stallman 認爲，軟體應該像空氣一樣，每個人都應有權利免費取得，而不應該受制於人，這樣一來，程式設計師豈不都失業了？不會的，Richard Stallman 說，他們可以做維護軟體的工作，畢竟不是每個使用軟體的人都會修改程式；只是，沒錯，程式設計師不會再賺大錢了，而他們賴以生存的將是提供服務、提供諮詢，事實上現在就有這樣的公司存在。更何況，他說，又沒有人強迫你要當個程式設計師...。

你會問，免費的軟體不提供支援，那有了問題怎麼辦呢？答案是，因爲大家都可以免費拿到程式碼，所以你不必擔心找不到人解決你的問題，會寫程式的人一大堆，不怕找不到人，這也是以後程式設計師的工作機會之一。而現在的軟體公司，收取了高額的費用，難道就提供較好的服務和軟體嗎？答案是，它們只會叫你再去買更新版的軟體，而更新版的軟體，通常意味著更高的價錢和更多潛在的 bug！

Richard Stallman 的觀念並不違反資本主義社會的運作原則，他認爲軟體並不像其他實質的物品，一般的工業產品都是經過一定的原料、製程才成形的，比如說麵包，你得有麵粉、機器、手工等等才能生產出麵包來；但是軟體不是這樣的生產法：它只要拷貝就可以了！他認爲，將軟體散布得越廣，越能證明該程式的有用，也就是對社會的貢獻更大，而如果有人嘗試要控制資訊、知識的散布，或者企圖阻止人們去分享它，那便是社會進步的破壞者，而若有人因此而致富，那更是不道德的，這樣的所謂著作權保護，不過是犧牲多數社會大眾的利益、妨礙文明發展，來造就少數人的財富罷了。而現在，我們的社會居然還大肆宣揚它，讚揚它，那就像我們付錢給某些破壞社會秩序的人一筆錢，拜託他們不要再阻礙社會進步一樣，世上還有比這個更荒謬的事嗎？

基於這樣的理念，Richard Stallman 硬是做出一番轟轟烈烈的軟體革命來，成立了“Free Software Foundation”(FSF)，更得到不少的支持，HP 等知名公司甚至提供機器給他們，連 *UNIX* 的東家 AT&T 都共襄盛舉。這個基金會的運作想當然是非營利性的，Richard Stallman 本人不領薪水，充當義工，裡面工作的程式設計師也都領夠過活的薪水而已，比起在外面工作，薪水確實不成比例，但是可以肯定的是，來到這裡都是心甘情願的，爲的只是一個理想。

這個理想有多少成果了呢？很多，其中 EMACS 是第一個產出，而 GNU 的 C Compiler (gcc) 更是被公認性能超越一般的商業產品，其他的軟體更是洋洋灑灑，不勝備載。目前，整個 GNU 缺的只是一個核心(kernel)而已，待核心[2] 完成，GNU 就可以完全取代 *UNIX*，成爲獨立而且完整的一套作業系統了！

GNU 的軟體在網路上可以隨意拿到，完全免費，一般的 ftp site 都會有個 GNU 的特別目錄，存放 GNU 的東西，他們的東西都會有一份聲明，告訴你，你可以跟別人共享它，但不可以以它牟利等等。爲了達到這個目的，防止有人利用他的程式賺錢，Richard Stallman 特別發明了所謂“copyleft”，以便有別於傳統的“copyright”，這樣一來，GNU 的程式可以永遠保持免費和自由取得，而不至於像有些 Public Domain 的軟體，再被用來牟利。你可以在每一個 GNU 的軟體內找到份文件。

除了 GNU 的軟體外，世界各地還有很多人在從事這樣的軟體革命，像 Linux 就是個例子，它是一套可以在 PC 上跑的 *UNIX*，也是免費的。另外值得一提的是，Richard Stallman 的基金會“FSF”是叫“自由軟體基金會”，“Free”指的是比“免費”更廣義的“自由”的意思，就是賦予使用者使用該程式的自由，當然，包括散布，複製等等。

另外，也許你會發現，網路上的免費軟體幾乎都是壓縮過的，這當然是爲了減少資料量的作法，也便於網路傳遞，但你可能發現，他們幾乎都是以“.gz”結尾，而不是“.Z”，也就是說，你得先拿到 GNU 的“gzip”程式，才能解開它們，而一般的 *UNIX* 指令“uncompress”是不能解“.gz”的檔案的。爲什麼不用 *UNIX* 的“compress”來壓呢？事實上，原來他們也是用 *UNIX* 的“compress”來壓成“.Z”檔，這樣的話，一般的使用者可以用現成的 *UNIX* 指令“uncompress”來解，問題出在“軟體專利”上。你可能不清楚，“compress”所用的 algorithm 居然是有專利的！也就是說使用“compress”來壓縮程式，再將此程式任意散佈出去的話，可能會觸犯著作權法！對於使用 *UNIX* 的人來說，這個問題當然不大，反正賣 *UNIX* 給你的人得去解決，但對“免費”的軟

[2] GNU 的核心叫“Hurd”，架在 Mach 上，已具雛形。

體來說，麻煩就大了，因爲它是被廣泛而且免費、自由散佈的！所以，後來大家就用"gzip"去壓縮免費軟體，而不用"compress"了，省得日後還挨告。

關於所謂"軟體專利"，又有一大串故事可講了，"The League for Programming Freedom"(LPF)這個組織，正在進行另一項以小搏大的歷史任務，它們倡言軟體不應該有專利，而正想盡辦法要推翻它，有興趣的讀者可以透過 FTP 到 ftp.uu.net:/doc/lpf 拿到相關資訊，或是透過 World Wide Web (WWW, W3) 到 http://www.lpf.org/ 取得，或是直接寄個 Email 到：lpf@uunet.uu.net 也可以。它們對抗蘋果(Apple)電腦公司的行動曾經引起廣泛的注意[3]。而目前，UNISYS 這家公司正宣稱它對 GIF[4] 檔案格式擁有專利權，LPF 對此也正採取行動對抗中[5]。以下是節錄自一份 LPF 發出的訊息：

> Although recent news has been hopeful on the user interface copyright front, the software patent front is getting worse. There are now over 14,000 US software patents; over 4,000 were issued in 1994, and the rate is increasing. Most programmers oppose software patents, but our opinions don't count unless we speak up. One way to speak up and be heard is to join the League for Programming Freedom.

在這個智慧財產權高漲的時代，也許我們都只是人云亦云，未曾認眞思考人類社會的眞正需求，了解 FSF 及 LPF 所要表達的訊息之後，或許我們還可以重新找回失去的反省能力。智慧財產權，究竟是文明進步的象徵，或是阻礙彼此發展，造就少部分人財富的制度，就有賴時間證明了。

3.5 編輯器

怕 *UNIX* 的人，有一半是被 vi 嚇跑的！眞的，如果你不喜歡用它，就不要用，沒有人說學 *UNIX* 就得要會 vi。但是筆者看過的 *UNIX* 入門的書裡面都把學 vi 說得很重要似的，眞是罪過！各位現在可以用 Openwindows 提供的 textedit 或 X-Window 的 xedit 來編輯檔案是很幸福的，想想在 20 多年前 *UNIX* 剛發展的時候，全螢幕編輯器是多麼偉大的事啊，筆者在學校用過 CDC Cyber 的編輯器，行編輯器哪！要 Delete 掉一個字元要先移動 cursor 對準目標再"D"下去，編個檔案跟作戰一樣，在這之前，還是用打卡的哩！所以各位也不必太責怪 vi 了，vi 的作者是誰呢？就是 Bill Joy！很耳熟？沒錯，就是 C Shell 的作者。他自己曾說，如果知道 vi 後來會這麼廣爲使用，他就不會寫它了。又聽說，我們現在看到的 vi 其實是不完整的，當初 Bill Joy 還在做改進，讓它更 user-friendly 一點，可是他的工作磁帶壞掉，他不曉得，等到發現的時候已經來不及了，又沒有備份，他只好放棄。

不用 vi？那有甚麼編輯器可以用呢？下面要介紹的編輯器，都不是視窗，可以在任何 Shell 裡面用。這裡不打算"教"它們的使用法，只是告訴大家有這些好 tool 可以用。先介紹"emacs"。

Emacs 不只是編輯器！除了編輯，你甚至可以在裡面玩 game，讀／送 mail。(它基本上可以取代掉 Shell。)它的作者就是 Richard M. Stallman，一個據稱是全世界寫程式寫得最多的人，各位現在可以自網路上取得很多很好用的免費軟體，他是最有貢獻的人了！

Emacs 的使用：指令"emacs"就是了。他有內建的 tutorial，可以幫助你上手，記住，要離開 emacs 時要打"Control-X + Control-C"，不要進得去卻出不來了！EMACS 已經被廣爲使用，據

[3] 蘋果電腦公司控告微軟 (Microsoft) 及惠普 (Hewlett Packard) 侵犯它在視窗系統"Look and Feel"的設計概念，這個案子已告敗訴，因此 LPF 也撤消對 Apple 的杯葛行動。

[4] GIF 檔是廣爲使用的影像儲存格式，它所使用的壓縮法則是大家熟知的 LZW 壓縮法，UNISYS 擁有此壓縮法的軟體專利，並據此對使用 GIF 檔案格式的軟體發展者要求給付權利金。

[5] 另外它對蓮花(Lotus)公司宣稱在使用者界面(User Interface)的著作權也在進行對抗之中。

稱，這個世界上的 *UNIX* 使用者，使用 emacs 和使用 vi 的人數約略相等。但，如同 vi，對一個 *UNIX* 新手來說，這樣一個功能健全的編輯器實在很難說好用，尤其是有 DOS 經驗的人。

有人用慣了 PC 上面的編輯器，像 WordStar，Borland 的 Turbo Pascal 等等的，一上 *UNIX* 還想保留原有的按鍵習慣，"Joe's Own Editor"是一個不錯的選擇。指令是"joe"，你不可能不會用它，它太簡單了，而且隨時提供你 on-line help，事實上如果你已經熟悉 Turbo 系列的 Editor，你不必擔心有任何不同，它的目的就是要讓 PC 的使用者可以在 *UNIX* 上使用同樣好用的編輯器。

還有一個選擇：PICO（指令："pico"）。它是一個很簡單的 Editor，沒有很花俏的功能，但對一般的編輯而言也夠了。

上面說的這些 editor，都不是"標準的 *UNIX* "[6]有的。如果你用的 *UNIX* 上面沒有，可以請你的 SA 安裝[7]。通常這些另外加裝的程式會放在 /usr/local/bin 下，你可以先看看這個地方。

3.6 你可能不知道

☐ 對於他一手創造的偉大系統，Ken Thompson 有沒有哪一點不滿意的呢？有一次他被問到，如果有機會再重新設計 *UNIX* ，他會做哪些改變？他說：我會記得在 creat() 的字尾加個 e . . . 。

☐ 你知道 SUN 公司的名稱由來嗎？當初史丹佛大學（Stanford University）利用 Motorola 68000 CPU 設計了一個電腦系統主機板，叫做"Stanford University Network Board"，簡稱"SUN"。史丹佛大學將此設計授權給商業公司，由於成本低廉，此一行動造就了日後 *UNIX* 工作站市場的蓬勃發展，而在激烈的市場競爭下，以母板名稱爲名，成立於 1982 年的公司"Sun"，一支獨秀，掌握了大部分市場，其成功的關鍵或許和它網羅了 SUN Board 的原始設計者之一的 Andreas Bechtolsheim，以及 BSD *UNIX* 的大將，Bill Joy 有關。

☐ 那個有點兒軟但卻賺很多錢的公司，終於弄出了一個 Windows NT。在此之前，各 *UNIX* 廠商早已動作頻頻，聯盟（COSE，OSF）不斷。 Novell 以 3.5 億美元收購 USL 後，各 *UNIX* 廠商才眞正有"團結"的跡象，爲的也是對抗 NT。之前， *UNIX* 由 AT&T 所擁有，而 AT&T 自己又發展、銷售 *UNIX* ，多少讓有關 *UNIX* 的商業行爲變得有點複雜。有趣的是， 發展 NT 的主要領導人 David Cutler，是 Microsoft 從 DEC 公司找來的，這個人在 DEC 時期曾帶領研發幾個作業系統，在離開 DEC 時正負責 VMS 的一個計劃。當初 VMS 在市場上輸給 *UNIX* ，沒能跨出 DEC 的機器，現在 NT 和 *UNIX* 又要對頭，不知道 Cutler 先生心裡面如何想？

另外，Windows NT"有點像，又不太像 *UNIX* "的特性，使得"NT 到底是不是 *UNIX* "的爭吵從不間斷。這種爭論到頭來可能要先解決"什麼是 *UNIX* "這個問題才行。而這樣的定義，讓筆者想到另一個命題：什麼是搖滾樂？這些問題，留給那些愛考究的人去傷腦筋，我們還是不要太爲難自己. . . 。

[6] 有"標準的 unix"這回事嗎？
[7] 或是自己安裝也可以，不過，對一個新手而言，這似乎太具挑戰性了。

想想看，如果有人同你說:
"只要你保證不拷貝給其他人的話，我就把這些寶貝拷貝給你用。"
其實這樣的人才是魔鬼;而誘人當魔鬼的，則是那些賣高價軟體的人。

by Richard Stallman

Chapter 4

你的殼 C Shell

還記得"殼"嗎？記住，從你 login 開始你就脫離不了它了。你的所有動作、指令，都跟它有關。*UNIX* 迷人的地方之一，便是她允許你用很簡單的方法重新設定你自己的環境。大部分的指令都不需有所謂的"設定"，但 Shell 的地位太重要了，所以它有一個特別的檔案來控制。藉由這個檔案的編修，你可以設定你自己的 *UNIX* 環境。以 C Shell 而言，這個控制環境的重要檔案我們在第一章說過，叫做".cshrc"，它應該在每個使用者的家目錄裡面，也就是說，每個人都有一個自己的".cshrc"，各自掌控個人的 C Shell 環境。

在你的系統管理人建立你的帳戶（account）時，他應該幫你準備好一份適用於你的環境的".cshrc"，你現在可以看一看你的家目錄裡面的".cshrc"，或是看看下面這個簡單的例子。

4.1 一個簡單的 .cshrc 例子

```
# @(#)Cshrc 1.6 91/09/05 SMI
######################################################################
#         .cshrc file
#         initial setup file for both interactive and noninteractive
#         C-Shells
######################################################################
# Set openwin as my default window system
set mychoice=openwin
#         set up search path
# add directories for local commands
set lpath = ( )
if ( ${?mychoice} != 0 ) then
    if ( ${mychoice} == "openwin" ) then
        set lpath = ( /usr/openwin/bin/xview /usr/openwin/bin $lpath )
    endif
endif
set path = ( ~ $lpath ~/bin /usr/local /usr/ucb /usr/bin /usr/etc)
#set lcd = ( )  #  add parents of frequently used directories
#set cdpath = (.. ~ ~/bin ~/src $lcd)
set noclobber
```

```
alias cd            'cd \!*;echo $cwd'
alias cp            'cp -i'
alias mv            'mv -i'
alias rm            'rm -i'
alias pwd           'echo $cwd'
set history=40
set ignoreeof
alias h            'history \!* | head -39 | more'
#alias ^L           clear
alias m            more
alias type         more
alias dir          ls
alias ll            'ls -la'
alias +w            'chmod go+w'
alias -w            'chmod go-w'
alias x             'chmod +x'
alias bye           logout
alias run          source
#alias lp1           'lpr -P1'
#alias help          man
#alias key           'man -k'
```

這是一個簡單的 .cshrc，除了以"#"開頭的行是用來註解的以外，每一行均有特別的意義，以下慢慢說明。希望透過這些說明，讓你學會基本的 C Shell 技巧，在使用 *UNIX* 時能夠清楚每一步驟背後所代表的動作，不再只是個只會打指令的 *UNIX* 打字員。

4.2 預定變數(Predefined Variables)

我們先回到在第一章所提到的問題：如何更改既定的系統提示號（system prompt）？ C Shell 的原始系統提示號是百分比符號："%□ "（% 後面有一個空格），但它是可以改變的。用的指令是"set"。例如，你希望提示號是"I Love UNIX >"，而不是（原來的）"% "。你可以下這道指令：

```
% set prompt = "I Love UNIX"
(注意有個 "=" 符號)
```

下面用一個例子顯示重設系統提示號的前後效果：

```
%
% set prompt = "<Is it true ?> "
<Is it true ?> echo "hello" (提示號變成新的設定)
hello
<Is it true ?>
```

除了提示號可以改變外，還有哪些東西可以由使用者自行定義呢？下面我們來看看這些稱爲"預設變數"(predifined variables)的作用和重新設定的方法。設定就是用"set"這個指令，其語法是：

```
% set variable = string
```

對照剛剛講的"改變系統提示號"的指令應該清楚："variable" 是你要設定的"預設變數"名字；"string"是你想要這個"預設變數"變成什麼樣子，一般來說是一個字串。不要忘了"="符號！

prompt　用來設定系統提示號，使用範例在上面說過了。

path　用來設定系統尋找指令的順序和路徑，跟 DOS 的 PATH 類似。例如，當你打下指令：

```
% more .cshrc
```

時，系統如何去找 more 這個指令，然後去做該做的動作呢？說得更誇張一點，假設你自己寫了一個程式，然後也把它叫做"more"放在你自己的家目錄下。當你下了"more .cshrc"指令時，系統到底是要執行系統裡的 more 還是你自己的 more 程式呢？答案是，看你的 path 這個變數怎麼定。假設你的 path 是這樣定：

```
% set path = (  /etc /bin /usr/bin /usr/ucb /usr/lib ~ .)
```

當你"more .cshrc"時，系統先到 /etc 底下找有沒有一個叫"more"的檔案，有的話就執行它，沒有的話就再到 /bin 下面去找，如果沒有，就再到 /usr/bin 底下找，依此類推。所以，一般我們會將常用指令所在的路徑放在前面，以縮短系統尋找的時間。而如果你常要執行"自己的"程式，那麼你可能會將你私人程式的所在路徑放在 path 前面一點的地方。偶而，你會發現你下一個指令，系統卻說"command not found"，或者，系統執行了某個不像你給的指令的動作，這時你應該要用 which 指令來確認一下，看看到底當時的 path 設定是不是你要的。

當然，你也可以指定全路徑名稱來使用某個指令，例如：

```
% /bin/who
```

就是指名要執行 /bin 這個目錄下的 who，而不要依照"set path = ..."的定義來找。

cdpath　這個變數的設定方法、格式和上面講的 path 一樣，都是一串路徑的集合，不同的是，上面講的 path 是當你下指令時，系統尋找指令的順序；而 cdpath，顧名思義，是專給"cd"這個指令參考用的。例如，你在使用 *UNIX* 時，常常要換到某些特定的目錄（假設是 ~/mail, ~/, ~/homework/math/linear, ~/homework/math/int），可以想見，你會常常打"cd ~/homework/math/linear" 這樣一大串文字，煩不勝煩，既浪費時間又容易打錯字，這時候你需要好用的"cdpath"：

```
% set cdpath = (~  ~/homework/math/linear)
```

這個設定告訴 C Shell："cd 目錄名字"到某個目錄時，優先在上面的路徑找同名的。所以像"cd ~/homework/math/linear"這樣長的指令，現在只要打"cd linear"就可以了！系統發現你有設"cdpath"，就沿著"cdpath"所定的路徑找"linear"：先在"~" （註："~"是表示你的家目錄，算是"簡寫"吧。）找，有的話就 cd 進去，否則就繼續到"~/homework/math/linear" 裡面找看有沒有叫"linear"的子目錄，系統在~/homework/math 之下找到 linear 這個目錄，二話不說就跳進去了。要注意的是，cd 這個指令永遠先在目前的工作目錄路徑(current working directory)下，尋找要跳進去的子目錄，然後才是依照 cdpath 所定的目錄去找。

所以，如果你當時所在的目錄下剛好也有個子目錄叫 linear，那就不會是跳到 ~/homework/math/linear，而是"目前路徑底下的 linear"了。

第二個要注意的是，cdpath 不像 path 非設不可，cdpath 不設的話，下 cd 指令時偶而比較麻煩，要給一大串路徑，但如果不設 path，那麼，每個指令都得要給全路徑(full path)，例如，"/bin/who"，"/bin/cat .cshrc"，要命的是，沒有人有閒功夫去記住每個指令的所在地！（雖然 path 也有預定值，但那通常是不夠的。）

另外，有一個有趣的現象你可以玩一玩。如果你用的是 SunOS，而且有設"cdpath"，而 cdpath 中不含有"目前路徑"（也就是說， cdpath 不含"."），而現在你要 cd 的目錄在目前的路徑，該目錄的權限對你而言是"不可執行"，換言之，就是"不可進入"。你用"cd 該目錄名稱"和"cd ./該目錄名稱"，會是什麼結果呢？你一定會猜：應該都會得到"Permission denied"的訊息。答案是，後者是會"./目錄名稱: Permission denied"，但是，前者卻會得到"目錄名稱: No such file or directory"！爲什麼會這樣呢？天曉得！大概要去問問 SunSoft 吧。

mail 這個變數是用來告訴系統，要去注意哪些檔案的變化，以便隨時通知。如果沒有設定這個變數，系統的預定值就是使用者本人的電子郵件信箱： /usr/spool/mail/"你的username"。

> 順道一提， *UNIX* 是將新收到的電子郵件放在 /usr/spool/mail 之下，在那之下的檔案，以使用者的 username 爲名，一人一個，例如，寄給"John"這個使用者的電子郵件，就放在"/usr/spool/mail/John"這個檔案內。使用者剛寄到的電子郵件就放到該檔案去，如果檔案已經存在了，就將新到的電子郵件附加在檔案的最後面，所以，如果該使用者一直不清理信箱的話，這個檔案會一直長大，一直到磁碟空間被佔滿爲止。又，這個"信箱"雖然是"一個"檔案，但可能包含成千上百封信，*UNIX* 的郵件系統有一套方法來分辨每封信。

也就是說，mail 的預設值是：

```
% set mail = ( /usr/spool/mail/使用者名字 )
```

所以，在你有新的信件到達時，系統把這個新的信件加到"/usr/spool/mail/你的userame"上去，然後（因爲這個檔案有了變化），系統便會告訴你："You have new mail."。

你會問，難道系統一直盯著這個檔，看它有沒有變化嗎？當然不是的，系統總有其它重要的事做，不可能隨時注意它。通常是每隔 10 分鐘去看一次，如果你覺得 10 分鐘太久，你也可以改變這個時間，只要在括號內"/usr/spool/mail/使用者名字"之前加上一個數字（以秒爲單位）就可以了：

```
% set mail = ( 300 /usr/spool/mail/使用者名字 )
```

這是叫系統每"300秒"去看一次信箱的意思。

除了郵件信箱，你也可以另外要求系統幫你再注意其他檔案。最常要被注意的檔案是一個叫"Message Of ToDay"的檔案，位在"/etc"之下："/etc/motd"。顧名思義，它是一個佈告欄，系統管理人會在這個檔案"張貼"公告，提醒使用者注意的事項。也因此，這個檔案是每個使用者每次 login 必定要看的。因爲太重要了，所以，使用者一 login，系統就會先將這個檔案(/etc/motd)，印在螢幕上，然後才是以前說的：讀取使用者的 .cshrc 、.login 檔。它是系統管理者最常用來公告注意事項的告示牌，比如說像："某天要停電，機器需要關機兩天，請各位注意時間的安排。"，等等需要通知每個使用者的訊息。系統管理者只要將訊息寫到 /etc/motd 這個檔案，這樣，每個使用者 一login，就可以看得到。然而，它只在有 login 動作時才會顯示，如果你已經 login 進到系統中，而在 login 之後，如

果"/etc/motd"又有了新的訊息，使用者就看不到這些新的訊息。這個漏洞可以用 mail 這個變數來補救：

```
% set mail = ( 60 /usr/spool/mail/使用者名字 /etc/motd)
```

這個設定是告訴系統：每隔 60 秒（一分鐘）就去看看電子郵件信箱以及 /etc/motd 這個檔案，其中之一一有變動就發訊息通知。例如，當系統在每一分鐘定時的觀察發現"/etc/motd"有所更動時，你將會看到："New mail in /etc/motd."的字眼出現在你的螢幕上。這樣你就知道該去看看"/etc/motd"的內容了。當然，如果你正在使用 vi 或正在跑其他程式，系統是不會打擾你的，上述訊息將在你再給下一個指令時才出現。

history　告訴 C Shell 要記住幾個"過去你給過的指令"。沒有設定這個變數時，他的預設值是 1， 也就是只記住你下過的上個指令。要它記住多一點過去的指令，就要設定它：

```
% set history = 40
```

這樣，C Shell 就會記住上 40 個指令，供你使用。當你要重複這些指令時，就不必再重打一大串英文字，只要給個號碼就行了。至於要如何使用，在後面我們會有詳細的介紹。

ignoreeof　*UNIX* 中，有幾個特別的控制字元，像 Control-C，Control-Z 等等，各有其特別功用，後面會介紹到。而其中，Control-D 是最特別的，它是"結束"（EOF, End Of File）的意思。記得在第一章提過，你 login 到 *UNIX*，其實是進到一個殼裡面，這個殼吃進你給的每個指令，然後再解釋給機器聽。如果你餵給殼的是"End Of File"，那就是告訴這個殼： 完了，的意思，要它自己結束；換句話說，那個殼就不見了。如果這個殼是最最外層的殼，那就跟"logout"沒有兩樣：你會被踢出 *UNIX*。按下"Control-D"來跳出 *UNIX*，是很方便，不必打指令，但相對的也太危險了，萬一不小心誤按了它，豈不是大災難嗎？一不小心就被踢出來了！有鑑於此，便有了 ignoreeof（ignore eof，忽略 EOF）這個變數：

```
% set ignoreeof
```

告訴系統："Control-D 不算數了"！當你設定了"ignoreeof"後，你按"Control-D"，殼會告訴你： Use "logout" to logout. 表示它現在不接受"Control-D"當"EOF"，要 logout 就要下指令，不能只按"Control-D"。

如果你覺得你很小心，不會誤按"Control-D"，希望能用"Control-D"來跳出殼，你可以不設"ignoreeof"：

```
% unset ignoreeof
```

這也是系統的預設值。

noclobber　前面講到的輸出重導向，可將螢幕的輸出導到一個檔案去，例如：

```
% man cat > file1   # 將 "man cat" 的結果放到 file1 去
```

有沒有想過，萬一 file1 是個原來就存在的重要檔案，那豈不毀了？輸出重導向固然好用，如果因此而有破壞性，副作用未免太大了！如果有這種顧慮，你可以用：

```
% set noclobber
```

來保護，它告訴系統：在輸出重導向時，如果要重導向的檔案已經存在，就不可以重導，以免破壞原來的檔案！而且，系統將回答你："檔名: File exists."，提醒你要導向的那個檔案已經存在。

同"ignoreeof"，如果你覺得你很小心，也不擔心會蓋掉已經存在的檔案 ，你可以：

```
% unset noclobber
```

這也是系統的預設值。

有時候"noclobber"的保護會很煩，你可以不要這種保護："unset noclobber"；或者，你只是要針對這次重導向，暫時取消保護，那只要在原來的重導向符號">"後面再加個"！"就可以了：

```
% man cat  >! file1
```

filec 這個變數讓你可以在打檔案名字時，不必打全名，只要打前面幾個字母，足以讓系統根據這幾個你打的字母，辨認出檔名即可。例如，你有

```
% ls -la
-rw-r--r--  1 robin      33054 Jul 18 15:05 csh.tex
-rw-r--r--  1 robin       2851 Jul 18 08:59 freeware.tex
-rw-r--r--  1 robin       6372 Jul 18 08:59 history.tex
-rw-r--r--  1 robin      14191 Jul 18 15:43 inout.tex
-rw-r--r--  1 robin       8145 Jul 18 08:59 internet.tex
-rw-r--r--  1 robin      38978 Jul 18 15:36 intro.tex
-rw-r--r--  1 robin        625 Jul 18 08:59 k-home.tex
-rw-r--r--  1 robin       1067 May 27 11:29 main.tex
```

這些檔案。下面三個指令中的<ESC>表示按<ESC>鍵，按下它後，檔案的全名自己會跑出來，不必再打後面的字母！[1]

```
% more c<ESC>  (相當於 "more csh.tex")
% vi inte<ESC> (相當於 "vi internet.tex")
% cat m<ESC>   (相當於 "cat main.tex")
```

注意第二個例子中，如果只打 in<ESC>，系統無法辨識是 inout.tex，internet.tex 或是 intro.tex，那麼<ESC>鍵就不起作用，因為 C Shell 在此情況下無法對應到唯一的檔案。

這個功能非常好用，有些人在替檔案取名字的時候，不敢取得太長，例如："This_is_a_FiLe_NamE_owned-by-John" ，因為檔名長，雖然能夠讓人一眼看出該檔案的特徵，但是打起檔名就會手指頭打結。有了"filec (Filename Completion)"這個功能後，就不必有這種顧慮了，因為你只要給前幾個字母再按<ESC>就可以，根本不必敲太多鍵盤，如果給的字母還不夠讓 C Shell 認出你要的檔案，反正邊加字母，邊按<ESC>也可以[2]。如果該目錄下只有一個檔案，那你連一個字母都不必給，只要按<ESC>鍵，那個唯一的檔案名字就出現了！

另外，在某些 *UNIX* 系統中，"filec"這個功能不只可以找出在目前目錄的檔案(的名字)，還可以對應到"指令"名字，也就是說，你只要打出某個指令的前幾個字母，再按<ESC>，系統便會在你定義的 path 中找到能對應的執行檔。比如

[1] 注意，如果你是用 Sun OpwnWin 的 cmdtool，這個功能可能會失效，建議你改用 xterm。
[2] 如果你用"Control-D"來代替 ESC，可以把有對應到的檔名全部顯示出來。

說在筆者使用的 HP 機器上，我只要打"make_f<ESC>"，C Shell 便自動幫我補成："make_floating"，它是一個筆者自己寫的小程式，位在筆者的"～/bin"[3]。

status 這個變數不是用來讓你設的，它自己會設定。每當你執行過一個指令，它就會根據執行的結果，設定這個變數，執行成功了，就把"status"設爲"0"；執行失敗，就設爲"1"。例如：

```
% date
Sun Jul 23 11:26:51 CST 1995
% echo $status
0
% datt
datt: Command not found.
% echo $status
1
```

這個變數平常使用的機會較少，但是在寫 C Shell script 時就會用到，用來判斷上個指令到底有沒有成功。

這是幾個比較重要的預設變數，簡單介紹到這裡。下面特別介紹"alias"和"history"的妙用，然後再介紹另一個重要的觀念：環境變數（Environment Variable）。

4.3 別名(alias)與歷史(history)

"alias" 和 "history"是 C Shell 剛出現時，不同於其他 Shell 的兩大特點。它是由 Bill Joy 在柏克萊大學時所創造，是屬於 BSD 系列 *UNIX* 的特色之一。後來再出現的 Shell 也大概都將這兩項功能放進去了。兩項功能大大的減少了使用者的打字次數，並允許使用者定義自己的指令，不必再煩惱不同的系統要記不同的指令。現在分別來看看他們的用法：

4.3.1 Alias

"alias" 的用法很簡單，舉個例子：

```
% alias moer more
```

這是說，"more"這個指令有個"別名"（alias）叫做"moer"，我打"moer"時跟打"more"的意思一樣，不要說打錯字，（"Command not found"），對於手指頭太快，常常打錯字的人來說，這個 alias 極爲受用。又如：

```
% alias ll ls -la
% alias m more
% alias dir "ls -la"
% alias lr "cd ~/homework/math/linear"
```

[3] 同理，如果在打出某個指令的前幾個字母後再按"Control-D"，那麼，C Shell 也會把所有對應到的所有指令都列出來。

這是說，我打"ll"時，意思就是"ls -la"。打"m"時，意思就是"more"，我懶得打"ls -la"和"more"這麼長！第三個就更完美了： 它是告訴 C Shell 一個新的指令叫"dir"，它的實際動作就是"ls -la"。對於習慣 DOS 的人來說，這意味著：你幾乎可以把 DOS 和 *UNIX* 的指令對應在一起，以後就不必再煩惱在不同的作業系統必須學習不同的奇怪指令了！而像第四個例子，表示你可以打"lr"來換到"~/homework/math/linear"這個目錄。相較於前面說的 cdpath，用 alias 更是簡潔。

所以，簡單的說，"alias"可以讓你定義一些自己的獨家指令。

4.3.2 History

history 是 C Shell 另一樣讓人不用不可的機制。在前面說預設變數（predefined variable）時談到，"set history = 40"是告訴 C Shell 說，要記住剛剛用過的 40 個指令，存起來，並予以編號。所以當你下"history"這個指令時，你會看到系統顯示出一連串剛剛打過的指令，前面還帶一個流水號。當你要再用到某個用過的指令時，你就不必再打整串的文字，只要給那個指令的流水號，前面再加個驚歎號（！）即可。history 的妙用尚不只這樣，下面介紹的一些符號與 history 的結合有令人驚訝的神奇功效，可以讓你不必再苦於冗長的打字夢魘。這些特殊符號是用來簡化你的打字的，其實是進階的用法，但初學者往往看得目瞪口呆，以爲是什麼了不起的絕招，崇拜、忌妒者有之，從此不敢碰 *UNIX* 的人也有，其實，說穿了就不值一毛錢，現在把他們列出來慢慢解釋，希望能讓讀者有一點認識，下次也能夠冒出這些奇怪的指令，先去嚇一嚇其他不明所以的人，之後，當然別忘了再教一教他們。

4.3.3 健步如飛

現在就來看看在 C Shell 裡面有哪些奇怪的表示法。

~ 就是"家"目錄(Home directory)！當只有一個"~"符號時，是指使用者本人的家目錄，如果"~"符號後面接著一個使用者名字時，就是指那個使用者的家目錄，例如：

```
% cat ~John/.cshrc
```

指的是把"John"這個使用者的".cshrc"顯示出來。

. 目前這個目錄名稱。相當於你打"pwd"這個指令後，出現的那一大串路徑(path)。

.. 母目錄，目前這個目錄的"上一層"目錄名稱。

$HOME 跟"~"同義，指的是使用者的家目錄。

$bage 你自己定義的變數名稱叫 bage 的。（你可以定義自己獨有的變數，細節在後面會提）

- 這個符號表示一個範圍，像"1-9"，"a-z"等等。例如：

```
% rm file[1-9]
```

是殺掉 file1, file2, file3, ..., file9（如果有這些檔的話）。如果範圍中的檔案不存在也沒關係，系統並不會有任何抱怨，除非你給的範圍中沒有一個檔案是存在的。

, 你會問：那如果是選擇性的範圍呢？例如，上例中，我們只要殺掉 file1, file2, file5, file6, file7，應該如何給指令呢？

```
% rm file[1-2,5-7]
   或
% rm file[1,2,5,6,7]
   或
% rm file[1,2,5-7]
```

這三個指令有完全相等的效果。

對於像 file1, file2, file3 這些在檔名中有很多字元重複，命名得很像的檔案，利用"-"，","來處理，可以很有效率。

* 這個符號用來匹配零個或多個字元，跟 DOS 差不多，只不過更廣義。例如：

```
% cp ab* ~/TMP
% cp ab*cd  ~/TMP
% cp *cd*  ~/TMP
```

第一個例子會將"以 ab 開頭的所有檔案"拷貝到"～/TMP"這個目錄，包括"ab", "ab.kk", "abcd", "ab-ab", "ab cd"。

第二個例子會將"以 ab 開頭而且以 cd 結尾的所有檔案"拷貝到"～/TMP"這個目錄，包括"abcd", "abbb.cd", "ab-cd", "ab cd"。

第三個例子會將"名字裡有 cd 的所有檔案"拷貝到"～/TMP"這個目錄，包括"cd", ".cd.mm", "abmmcdpp", "ab cd"。注意，在 *UNIX* 中，你可以將檔名任意變化，即使像上述的"ab cd"這樣，檔名含有空格的名字，也是可以的！甚至，可以有控制字元，你可以試試下面的例子：

```
% touch ^Ewhat? (^E：Control-E，不是 shift-6-shift-E)
% ls -la *what*
    ....
% rm -i ^Ewhat? (^E：Control-E，不是 shift-6-shift-E)
```

這個例子讓你清楚的知道 *UNIX* 的檔名能耐！

? 這個符號用來匹配單一字元。例如：

```
% cp ??cd? ～/TMP
```

會將像"zzcde", "a-cdb", "2.cdm", "cd"這些檔案，（cd 之前有兩個字元，之後有一個），拷貝到"～/TMP"這個目錄。請注意與上面第三個例子的差別：像"cd", "cd.mm", "abmmcdpp" 這些雖屬於"*cd*"，並不符合"??cd?"的規格 ！

! 在 C Shell 內，驚歎號"！"用來"回憶歷史"(history)，兩個驚歎號表示"剛打過的那一個指令"。比如你剛剛才打過 "ls -la" 來看檔案清單，但是檔案太多了，而你只是要看".c"的檔案，一下子沒看清楚。你當然可以再下一次 "ls -la *.c" 來看，但也可以就打"!! *.c"，在這裡，"!!" == "ls -la"。當指令很長的時候，這招很管用。

前面提到 history 可以記住最近的 40[4]個指令，"!!"表示"上一個"，那其它 39 個呢？就必須用號碼來代替了。你可以先用"history"這個指令來看整個歷史清

[4] 看你的 set history = xx 的 xx 是多少而定。

單(history list)，從中看出你要的那個(行)指令，再用它的流水號來代替。例如，你在不久前剛剛打過下面指令 ：

```
% cc -g -o proc conf.h proc.c f1.c f2.c f3.c -lm -lX11
```

然後你修改了 f1.c，現在要再 compile 程式一次。你可以先看 history，找出流水號，再重新執行：

```
% history
      ...
    17 cp proc.c proc.c.bak
    18 more f1.c
    19 more f2.c
    20 cc -g -o proc conf.h proc.c f1.c f2.c f3.c -lm -lX11
    21 ls -la *.c *.h
    22 vi f1.c
% !20 （驚嘆號加歷史號碼）
```

這裡打"!20"跟打"cc -g -o proc conf.h proc.c f1.c f2.c f3.c -lm -lX11"是完全一樣的！可以比較一下，利用歷史，可以省卻多少按鍵，可以節省多少時間，可以避免多少打錯字...。

!-1 倒數第一個指令，（也就是前一個指令，跟"!!"完全同義。）

!-2 前面打過的指令裡面，倒數第二個。例如：

```
% history
    78  cxtk24 csh
    79  set prompt = "<Is it true ?> "
    80  echo "hello"
    81  so
    82  h
    83  pwd
    84  ls -la
    85  more csh.tex
    86  more test.cst
    87  vi verb
% !-2
```

"!-2"相當於第 86 號指令："more test.cst"。

!mo 歷史清單裡面，倒數最近的一個以 mo 開頭的。看個例子：

```
% history
    78  cxtk24 csh
    79  set prompt = "<Is it true ?> "
    80  echo "hello"
    81  so
    82  h
    83  pwd
    84  ls -la
    85  more csh.tex
    86  more test.cst
    87  vi verb
% !mo
```

以這個歷史清單而言，86 號 ("more test.cst") 是你下"!mo"眞正執行的指令。

!?xyz? 歷史清單裡面，倒數最近的，含有"xyz"這些字的指令。看個例子：

```
% history
    78  cxtk24 csh
    79  set prompt = "<Is it true ?> "
    80  echo "hello"
    81  so
    82  h
    83  pwd
    84  ls -la
    85  more csh.tex
    86  more test.cst
    87  vi verb
% !?csh?
```

以這個歷史清單而言，85 號 ("more csh.tex") 是你下"!?csh?"眞正執行的指令。

!78 歷史清單裡面的第 78 號指令。上例中即是"cxtk24 csh"。

^ 這個符號我們叫它做"modifier"，用來修改剛剛打過的某一個指令的某一些字。比如說，你剛剛下了這個指令：

```
% cat file1 file2 file3 file44 file5 file6 file7 > file8
```

你的意思是想把 file1 、file2 、file3 、file4 、... file7 連接成一個檔案叫 file8，可是你打錯字了，將"file4"打成"file44"，多打了一個 4。重打嗎？指令那麼長，太累了！這時候你可以利用^，把 44 改成 4：

```
% ^44^4
```

這個指令的意思是說：重複上個指令，但是把上個指令的"44"改成"4"。

:s 前一個例子說可以用^來修改前一道指令；如果是要修改歷史清單裡的某一個指令（不只限前一個），而不一定是剛剛那一個，也有方法，就是用":s"這個整容指令。"s"是"search & replace"的意思。以剛才那個 "cc -g ....."爲例。你想把"conf.h"改成"conf.c"。你只要打下面這個指令：

```
% !20:s/.h/.c
```

就可以了。它的意思是說：在歷史清單裡，重複第 20 號指令，但是把指令中第一個".h"改成".c"。

注意，這裡的代換，只針對第一個出現的".h"做更改，如果有兩個以上，則要多加一個"g"在"s"前面，用":gs"。再看同一個例子，如果要把所有的"proc"全都改成"prog"：

```
% !20:gs/proc/prog
```

因爲 proc 在整個指令字串中出現了不只一次，而且你是要換掉所有的"proc"，所以必須指明是全部的"proc"，而不只是第一個。在上一個例子中，因爲".h"只有出現一次，所以可以想見，用"s"或"gs"都一樣。"g"是"global"，全部的意思。

: 冒號“：”用來挑選清單指令裡，某個特定的字。像剛才說過的，可以將歷史清單裡的某個指令，整個叫出來修改、執行，但如果只是要將某個清單指令裡的某個字叫出來呢？以下面的例子來說，假設有個指令在歷史清單裡的編號是26：

```
26 diff ./hw.2  ~/homework/math/linear/hw.1
  ( 0      1               2              )
```

現在你要列印“～/homework/math/linear/hw.1”。你當然可以不厭其煩地下這個指令：

```
% lpr ~/homework/math/linear/hw.1
```

但如果你會現在介紹的“:”的歷史功能，事情就簡單了，因爲你要的字樣已經出現過在歷史裡面：

```
% lpr !26:2
    或
% lpr !26:$
```

“!26:2”是表示：歷史清單裡編號 26 的那行指令的“第 2 個字”。注意的是，算的時候是從 0 算起，也就是說，指令本身是編號 0。所以，“diff”是算 0，“./hw.2”是算 1，“～/homework/math/linear/hw.1”是算 2，注意，只要有空格或像“>”，“&”這些特別意義的字元隔開就算是另一個字了。

而另一種表示法則是用“$”這個符號代表指令行的“最後一個”字，例如，“!26:$”表示歷史清單編號26的那行的“最後一個字”（以此例來說，就是編號 2）。相對的，也有特別的符號表示“第一個字”，那就是“^”，以這個編號 26 的來說，“!26:^”就是指“./hw.2”這串字。

如果要的不只是歷史清單指令中一個字，也可以用範圍來表示。例如，要列印上述歷史清單編號 26 的例子裡的兩個檔案：

```
% lpr !26:1-2
   或
% lpr !26:1-$
   或
% lpr !26:^-$
   或
% lpr !26:*
```

“*”是全部的意思，利用“$”和“^”的場合很多，這屬於“regular expression” 的討論範圍，在這裡先不做深入的講解，在後面有專門的章節再做介紹，有興趣的讀者也可以找相關書籍深入學習。

!$ 前一個指令裡面的最後一個字。以“cc -g -o ...”那個例子來說，就是“-lX11”。“cat ......” 那個例子裡，則是“file8”。例子：

```
% ls file* unix.txt
file1        file2         file3         unix.txt
% ls -la !$
-rw-------  1 KMJNS        320 Jul  8 16:23 unix.txt
```

學到這裡你會發現，事實上，“!$”和“!!:$”，和“!-1:$”是一樣的（記得嗎？“!!”就是“前一個指令”的意思）。

!* 前一個指令裡面的所有參數（不含指令本身）。以"cc -g -o ..."那個例子來說，就是指 "-g -o ..."這一大串。例子：

```
% ls file* unix.txt
file1        file2         file3         unix.txt
% ls -la !*
-rw------   1 KMJNS        102 Jul  8 12:02 file1
-rw------   1 KMJNS        102 Jul  8 12:02 file2
-rw------   1 KMJNS        102 Jul  8 12:02 file3
-rw-------  1 KMJNS        320 Jul  8 16:23 unix.txt
```

> 輸出資料重導向，Output Redirect。像剛剛提到的一些 command，他們的輸出都是到 monitor 上來，包括你自己的程式在內。但有時候你並不想這些 output 跑出來干擾你（比如你跑一個 background 程式，而它會 output 一些 message），你想把這些 output message 先收集起來，有空再慢慢看，這時，你可以用">"來做：

```
% my_program > message
```

如此一來，my_program 的所有輸出就會留在 message 這個檔案裡了。又比如前面的 "cat .... > file8"的例子。本來用 cat 這個指令會把 file 的內容 show 在螢幕上，用">"可以把原來 show 在螢幕上的東西重導到一個 file 去。

>& 標準錯誤輸出重導。在前兩章曾提到 *UNIX* 的標準輸入(standard input)與標準輸出(standard output)。事實上還有另一個重要的裝置叫"標準錯誤"(standard error, stderr)。前兩章已經說過標準輸入及標準輸出的作用了，那標準錯誤又是甚麼東西呢? 先試試這樣一個指令:

```
% ls -la /etc/*rc* /etc/ABCDEFG > message
/etc/ABCDEFG not found
```

你也許還沒有發現奇怪的地方: "/etc/ABCDEFG" 這個檔案不存在，所以系統回覆"/etc/ABCDEFG not found"，不是很正常嗎?再看看有個">"符號，就有些奇怪了。照前面的說法，有了">"符號後，輸出到螢幕的東西不是應該被重導到"message"這個檔嗎?怎麼還有"/etc/ABCDEFG not found"這些字跑出來?(它們不是應該在"message"這個檔嗎?)

沒錯，如果"`ls -la /etc/*rc* /etc/ABCDEFG`"這個指令有正確的結果的話，它的輸出是會被放到"message"這個檔，問題是上述指令的後半段根本沒有執行（由於"/etc/ABCDEFG" 這個檔案不存在，造成指令無法完成），所以也就沒有所謂輸出。

那"/etc/ABCDEFG not found"是甚麼呢? 它是"錯誤訊息"，是"系統的標準錯誤"，不是指令的"正常輸出"，所以沒有被重導到"message"這個檔案!

事實上，這些"錯誤訊息"是要被放到系統的"標準錯誤裝置"的，而所謂"系統的標準錯誤裝置"，在一般情況下，跟"標準輸出裝置"一樣，就是你的螢幕。

所以啦，當執行程式、指令的過程中發生問題，系統的錯誤訊息也是放到螢幕的。如果你連這些錯誤訊息也不要它們跑到螢幕前來，而希望把所有的輸出(包括指令的執行結果、系統的錯誤訊息輸出等等有的沒的)，都放到另一個檔案時，光用">"還不夠，因爲它只對指令的執行結果輸出做動作，對那些"錯誤訊息"並沒有作用。要"對付"那些討厭的"錯誤訊息"，要用">&"這個符號，它使得"重導" 的對象包含標準輸出(stdout)及標準錯誤(stderr)。換言之，把螢幕上的輸出都一網打盡了!

試著比較下面這幾個指令的結果，會更有概念:

```
% ls -la ~/.cshrc no-such-file > only-list
% ls -la ~/.cshrc no-such-file > only-list &
% ls -la ~/.cshrc no-such-file >& list-and-error
% (ls -la ~/.cshrc no-such-file > only-list) >& only-error
```

比較上面各個指令的輸出檔案，弄清來龍去脈之後，對於輸出結果的處理，將有很大幫助。注意最後一個指令我們用"("和")"來將一串指令先括起來當成一個先處理的部分，然後">& only-error"是對這個被括起來的一長串指令有作用。

另外值得一提的是，你會發現前面說的輸出處理都是講怎麼把那些輸出存起來，但有時候我們只是不想讓那些輸出跑到螢幕來干擾我們的工作，甚至連輸出結果也不關心，連存下來都嫌它們佔空間(特別是輸出的東西太多又沒用時)。有沒有方法可以把這些東西扔掉呢? 當然有的，那就是把輸出改丟到一個特別的檔案: /dev/null 。

/dev/null 是一個很特別的檔案，說它是檔案，倒不如說它是"黑洞"，所有被重導到這個檔案的東西都會消失得無影無蹤。把上面的例子中，"only-list"等輸出檔案換成"/dev/null"，再執行一次，比較一下結果就會明白了!

< 輸入資料重導向，Input Redirect。同理，輸入也可以重導向，比如你要寄個 mail 給某人，本來你打"mail somedoby" 後，你就會進到 mail 裡讓你寫信，然後再送出。你也可以先寫好了內容放在 letter 這個檔案裡，然後直接打：

```
% mail somebody < letter
```

就好了。本來要由螢幕一個一個輸入的東西，經由"<"就可以變成由檔案輸入了。

& 這個符號"通常"用來把一個 program 丟到 background 去跑。例如：

```
% xterm &
```

另外，也用在 C Shell 的 "remembered pattern"，這又是有點兒高竿的人才會用的，這裡不多說，以免嚇到初學 *UNIX* 的人。

^Z 前面說用"&"符號可以把 program 丟到 background 去 run，但當一個程式或指令已經在執行，如何再把它放到 background 呢? 用"Control-Z"及"bg"("background"的縮寫)。當你在程式執行當中按下"Control-Z"後，系統接收到"暫停"的信號，就會把跑到一半的程式冰凍起來，然後你會看到"Stopped"的字眼，告訴你該程式已經被暫停了。接著如果你再下"bg"這個指令，系統就知道要把那個暫停了的程式丟到 background 去繼續執行。當然，如果你冰凍了好幾個跑了一半的程式，你就得明白告訴系統你要丟到幕後去跑的程式是那一個，這時你可以用"jobs"指令先看看現在在這個殼裡面的程式都是些什麼狀態，例如：

```
% jobs
[1]  - Stopped              man cat
[2]  + Stopped              vi k
[3]    Running              xterm
[4]    Stopped              cat *tex
```

由這個例子看到，在這個殼裡面現在有兩個被冰凍起來的程式，第一個是"man cat"這個指令，還沒"man"完就被暫停了；第二個是"vi k"，也是沒有結束

就被冰起來；第三個是"xterm"，這個程式正在跑（執行）當中，所以狀態是"Running"；第四個也是還沒跑完就被冰起來的工作（"job"）。

另外看到在"[1]"後面有個"-"符號，"[2]"後面有個"+"符號，那是說：2 號被停的程式是比較後來被冰起來的，等一下如果要再起來跑，它應該優先，當這個優先的程式（有"+"符號的）起來後，下一個就是這個有"-"符號的1號程式排在優先（變"+"符號）。所以，如果你下"bg"這個指令，因爲沒告訴系統到底要把誰丟到幕後去跑，系統就把那個帶著"+"符號的拿出來，丟到幕後去繼續跑；然後，帶著"-"符號的就會改成帶著"+"符號，排在最前面等著被叫醒，而四號的"job"遞補成帶"-"符號（也就是說："+"符號表第一順位，"-"符號表第二順位，沒有符號的是其它）。如果要直接指定是哪個背景工作，也是可以，就在"%"後面加個號碼：

```
% bg %1
```

這樣就是指定要 1 號程式繼續在背景跑。

跟"bg"相反的還有一個"fg"指令，用來將一個被冰凍起來，或是在背景跑的程式，拿到"前景"（foreground）。

用"%"來指定背景工作除了用編號外，還可以用其它方式：

%	只給"%"，或是"%+"，或"%%"都是指那個有"+"號的背景工作。所以，"%-"指的是誰，你應該知道吧。
%字串	在"%"後面緊接著一個字串，指的是所有背景工作中，以該字串爲首的指令行，這裡說的"字串"可能是一或多個字母。例如，上面那個例子： % %v 或 % fg %v 表示把 2 號工作，"vi k"，拿到"前景"來跑。
%?字串	在"%"後面緊接著"？"，然後再接著一個字串，指的是所有背景工作中，含有該字串的那個 job。例如，上面那個例子： % bg %?tex 表示把 4 號工作，"cat *tex"，拿到"背景"去跑。

有時候你要把某一個被冰起來的程式或已經在系統背景跑的程式殺掉，也可以：

```
% kill %2
```

這是把 2 號程式殺掉的意思，不要它繼續跑。這個"2"就是用"jobs"看到的那個號碼。

如果要把在系統背景跑的某一個程式再帶到幕前(foreground)，用:

```
% fg %3
```

這個指令是把原來在 background 跑的 3 號程式再帶到 foreground 來執行的意思。注意在用"fg"和"bg"時，號碼之前要加"%"符號。

"kill"指令中的號碼可以有兩種意思："kill 號碼"的號碼，不加"%"符號的號碼，是 process id，就是用"ps"看到的那種；"kill %號碼"的號碼，有加"%"符號的號碼，是 job number，就是用"jobs"看到的那種。事實上，你把它丟到背景去的那些程式，也都有他們相對的程序號碼（process id），你也可以用 ps 指令看得到，所以，要殺掉它時，用"kill PID"這種方法也是可以的。只是這個號碼和用"jobs"指令看到的，是不一樣的。同樣是殺，方法有兩種就是了！

附帶一提，有時候有些程序（process）光用"kill"還沒辦法殺掉，這時候你可以試試加上"-9"這個選項: "kill -9 號碼"或"kill -9 %號碼"。

; 這是用來分隔兩個指令的符號，用在你"三步併作兩步"的下指令時。下面是一個例子：

```
% cd ~ ; cp .cshrc .cshrc.backup ; cp .login .login.backup
```

這裡，在同一行指令中你給了三個動作，他們依序完成，就像你分三次打指令的結果一樣。不同在哪裡？ history 會少一點囉！

| 管路。這是 *UNIX* 另一樣絕招，它可以讓你"暗渡陳倉"。怎麼說呢？它是一個兩頭空的管子，一邊來，另一邊去。舉個例子：

```
% ps -auxww | grep verilog
```

前半段是用"ps -auxww" 來看目前系統目前的工作狀態（有哪些 process 在 run），這樣列出來可能有幾十行，而你想知道的只是有關 verilog 的資料而已。你可以用">"把 ps 的結果重導向到某個暫存檔，再慢慢看，慢慢找。而利用"pipe" 你可以省掉產生暫存檔的動作：把某個指令的輸出結果送進管子的一頭（"|"的左邊），另一頭（"|"的右邊）接給另一個指令。這樣做的意思是：把 "|" 符號前的指令的結果，給"|"符號後的指令當輸入。所以上面的例子"ps -auxww" 的輸出中，含有 verilog 的那些行就給抓出來了。

"more"常用來承接其它一些指令的輸出，如："ls -la | more"與 DOS 的"dir /p" 就有同樣功效。

\ 取消別名。前面談到取"別名"（alias） 的妙用，它可以讓你更改指令名稱。但有時候，你想"暫時"取消某個 alias 。比如，通常我們會定義殺掉檔案時，要先詢問(inquire)一下，確認是不是眞的要殺掉，以免誤殺檔案，所以有 "alias rm rm -i"這樣一行在你的".cshrc"中；所以每次你用 "rm" 去殺檔案時，你都得回答"y"或是"n"。這樣一來，你殺檔案時就多了一層保障。但有時候這個保護就很囉唆，比如說你百分之兩百確定要殺掉一大堆東西（一大堆檔案、一整個目錄，等等），要一一去回答 "y" 會累死人，這時候就可以把"rm"的別名先"暫時"取消掉："\rm"，就是在指令前面再加個"\"就可以了。（簡單講，有加"\"的，是原名，沒有加的，是別名。）

另外一個方法，則是利用"yes"這個點頭指令以及先前說的"pipe"。例如，你有一整個目錄（假設叫 "garbage"）很肯定要全部殺掉，而你又有"alias rm rm -i" 這樣一個別名，下面兩個指令都可以達到目的：

```
% \rm -rf garbage
% yes | rm -r garbage
```

第一個指令告訴 C Shell ，回復"rm"的原始功能，將"garbage"以下的東西都殺光。第二個指令告訴 C Shell ，將"yes"這個指令的輸出送給"rm -r garbage"當輸

入；因爲後者會問很多"yes or no"，就讓前者去填這些回答。不要懷疑，眞的有一個指令叫"yes"，它什麼都不會，只會回答"y"！

好奇嗎？是不是也有"no"指令呢？沒有"no"這個指令，但是，如果你堅持要"no"的回答，只要說"yes no"就可以了：

```
% yes no | rm -r garbage
```

事實上，"yes"這個指令是將跟在"yes"後面的字串無限次輸出，試試看這個：

```
% yes 'I Love UNIX'
```

別忘了用"Control-C"來停掉"yes"指令的輸出，否則，它會一直"yes"個不完！

比較重要的預設變數都在上面介紹了，要特別一提的是，你可以自己定義其他的變數，例如：

```
% set number = 1
```

是定義一個新變數"number"，其值爲1。或：

```
%set beta = "Beta"
```

是定義一個新變數"beta"，其值爲"Beta"這個字串。另一個問題是，要怎麼看某一個變數的值到底被設成什麼呢？用"echo"指令，例如：

```
% echo $beta
Beta        (系統回覆)
```

注意，在 C Shell 中，我們用"$變數名稱"來取得該變數的內容。

或者，你可以用"set"指令來看所有已經（用"set"）定義過的變數有哪些，再看你所要了解的變數是不是定義對了。

又，什麼時候需要自己再定義這些本來不存在的變數呢？大概有兩個場合：一是寫 Shell Script 時，在做 Shell programming 時，常常必須用到變數，就像一般人用任何程式語言寫程式需要用到變數一樣。（不要忘了，C Shell 本身也是一個獨立的程式，它也有自己的語言，很像 C 語言就是了）。二是偷懶時，例如，因爲某些理由，你時常要打"/usr/local/lib/a/b/c"這樣一大串字，你可以自己定義：

```
% set pp = "/usr/local/lib/a/b/c"
```

這樣一來，你便可以用"$pp"來代表"/usr/local/lib/a/b/c"，不必再打一大串字了！

大致介紹完在 C Shell 裡頭常用到的符號和歷史清單用法之後，下面要談的是另一個在 C Shell 裡面，地位同樣重要的觀念：環境參數（Environment Variable）（或叫"環境變數"）。

4.4 環境變數(Environment Variables)

先看看在你的".cshrc"或".login"這兩個檔案裡面，有沒有一些行，內容格式大概是這樣子：

```
setenv ABCD "What-is-This:And-This"
```

那是用來定義環境參數的(environment variable)。上面的"ABCD"和引號內的文字只是用來說明格式，沒有其它意義。

"環境參數"一般是讓"相對應的程式"來"參考"用的。這裡所謂"相對應的程式"是指某些程式在執行時，需要有人指引明路。例如在前面章節提到的，你在 C Shell 裡面下一個指令，C Shell（注意， C Shell 本身也是一個程式，就是 /bin/csh）怎麼去找到你所要的指令呢？就是 PATH 這個環境參數指引 C Shell 去找的。而 PATH 這個環境參數的設定是透過 "set path = ... "來"間接"完成的。也就是說，你 "set path = ... "之後，PATH 這個環境參數就跟著設定好了。在這裡，PATH 這個環境參數就是執行任何程式的時候都要參考的。

你大概要問：前面說的"預設變數"（C Shell predefined variables）和這裡要說的"環境變數"（environment variables）有何不同呢？基本的差異在於，預設變數，只在它被定義的那個殼（Shell）有用，範圍僅及於該"殼"；而環境變數的效力範圍則包括定義該環境變數的那個殼，以及在那個殼之下再被啓動的程式，這裡說的程式當然包括 C Shell，殼，本身。傳統上，我們用小寫字來定義預設變數，而環境變數則大都用大寫的字。而定義環境變數的指令也和定義預設變數（用"set"指令)不同：

```
setenv ENVVAR string
```

注意，變數名稱後面沒有等號，這是跟"set"指令不一樣的地方。

下面我們來看看有哪些環境變數是已經由系統幫使用者定義好的，藉由說明的過程，邊說邊讓讀者了解環境變數的運作，以及它和預設變數的分別與關聯。

HOME 這個環境變數的值就是你的家目錄的全路徑名稱。例如："/home/ds1/johnsmith"。它在你 login 時，系統就由"/etc/passwd" 這個檔裡頭去找到你的家，然後把"HOME" 這個環境變數設成你的家目錄位置。這個環境變數會影響另一個預設變數（對，就是前面說的那種預設變數），"home" 。前面講預設變數時說漏了"home"。"home"這個預設變數是由"HOME"這個環境變數來間接設定的。

PATH 這個環境變數的值跟前面講的"path"這個預設變數一樣。在一開始，你 login 時，系統會先定義"PATH"成"/bin:/usr/bin"，而當你定義"path"時，"PATH" 就自動改成和 path 一樣[5]。你又要問啦，爲什麼要有兩份同樣內容的變數來作同樣的事呢？不要忘了，環境變數的效力可延伸到該殼以下的程式，所以當你改變了"path"這個預設變數後，"PATH" 若是不跟著動，那麼在那個 Shell 之下跑的程式就沒法用到"path"所定義的東西了，而藉由"PATH"，他們可以同樣有"path"的資訊，同樣可以找到要執行指令的位置。到這裡我們已經說過很多次所謂"在那個 Shell 之下跑的程式"，到底還是舉個例子比較清楚：假設你現在是在某個Shell，你可以先設定一個預設變數，叫"abc"，然後在那個Shell裡面再執行一次 C Shell，看看在這個"子殼"裡有沒有"abc"這個變數：

```
% set abc = "1234"  ------> 定義變數 abc 成 1234
% echo $abc         ------> 看變數 abc 的值
1234                ------> 果然是 1234
% /bin/csh          ------> 再進一次 C Shell，也就是，目前這個殼生出
                            另一個子殼，進到這個子殼內。
% echo $abc         ------> 看看在子殼中的 abc 是不是定義了？
                    ------> 沒有定義，子殼沒有繼承母殼的 abc 變數！
% exit              ------> 離開子殼，回到母殼。
% echo $abc         ------> 再看變數 abc 在母殼的值
1234                ------> 在母殼眞的有定義，沒錯。
```

[5] SunOS 是這樣，其它系統可能不一樣。

上面是一個很簡單的例子，可以看出用"set"所定的預設變數"abc" 並沒有在其後的程式 (/bin/csh) 中生效。你可以同樣用這種簡單的驗證法，將第一行的指令換成：

```
% setenv abc "1234"
```

來驗證前面章節講的預設變數和這一章節講的環境變數，體會他們的不同。

TERM 它用來設定終端機型態。如果你的終端機是直接接到主機的話，系統會自動將此變數設好，但如果你是透過網路登錄到機器，那你可能要設定它，以免使用像"vi"或"more"這些跟終端機形態有密切關係的程式時，出現螢幕不正常捲動的情況。一般，如果要設的話，大多設成：

```
% setenv TERM vt100
```

"vt100"是大部分系統都有定義的終端機形態[6]。

LOGNAME 有的 C Shell 用"LOGNAME"，有的則用"USER"。這個變數就是使用者的 username，也就是你 login 時的那個使用者名稱。跟"HOME"一樣，在你登錄進系統時，系統就由"/etc/passwd"中取得資料，自動設定好了。

好，現在你應該知道環境參數是做什麼的了，至少也有一點點概念。上面講的幾個環境變數都有個 default （預設、內定）值，一般而言並不需要再定義。例如"HOME"這個環境參數，它的內定值是你自己的家目錄，就是"～"。又比如， USER（或是 LOGNAME）這個環境參數，它的內定值是使用者本身的 username。他們是天生有用的，先有個預設值，免得 user 一進到 *UNIX* 裡面什麼都不對勁。這些環境參數早就存在了（像剛剛提到的 PATH，HOME，USER），也有 default 值，但有時候並不適用，像 PATH 就是，每個人用的軟體、程式放的地方不一樣，所以每個人的環境參數 PATH 就不會一樣，因此就必須重新定義。

剛剛提到的 PATH、HOME、USER 等等，是本來就有的，頂多再定義一次。但你也可以再定義自己的環境參數做自己特殊的用途。所謂特殊用途，有幾個意思，可能你安裝了某個軟體，它需要有個環境參數來輔助，所以它會規定你要定義某個環境參數，像 cxterm，它是中文的 xterm，可以在裡面使用中文。它需要你跟它說中文字形、輸入方法，等等是在甚麼地方，所以它需要一個環境參數(HZINPUTDIR)來告訴它中文輸入法在什麼地方。

你會說，哎呀，*UNIX* 怎麼那麼笨，把程式要用的資訊都放進程式內不就好了？你說對了，*UNIX* 就是把環境參數定在程式內。比如剛剛說的 HZINPUTDIR，這個字是固定的，cxterm 就靠它來認路。它程式內只定死 HZINPUTDIR 這個字，至於你要把 HZINPUTDIR 定義成哪裡是你自家的事，你愛把中文輸入法放到哪裡隨你高興，你哪天愛玩，自己創造了獨門的輸入法也可以，只要把 HZINPUTDIR 這個環境參數改一下就可以了，不必再重新編譯(re-compile) cxterm 的原始程式。這樣比較有彈性不是嗎？

除此之外，你也可以只爲了好玩，自己定義特別的環境參數。比如說，你覺得每次要打某一大串路徑或名稱很麻煩，你可以定一個環境參數給它，然後你就可以只用"$環境參數"來代替了，就像前面說的預設變數一樣。

但是要注意，不要定義一個和前面列的特殊用途的環境參數名字一樣的變數，以免系統或程式搞亂了。

在".cshrc" 內你可能還會看到幾個更特別的環境參數，這裡要說明一下：

[6] 附帶一提，終端機形態的資訊是放在"/etc/ttytype"中，它又跟"/etc/termcap"有很密切的關係，如果有極大的興趣，可以去研究一下，有點複雜，在此不作討論。

MANPATH 這個環境參數定義了 manpage 的搜尋路徑。manpage 指的是 manual pages，就是 *UNIX* 的線上查詢(on-line help)資料，就是你打"man 指令"之後出現的那些東西。標準的 *UNIX* 線上查詢資料是放在："/usr/man/man?" 裡的，"?" 表示 1, 2, 3, . . . , l, n 等。裡面的資料都是特殊格式(nroff/troff)的文件，依類別放在不同的目錄下，如 "cat"、"cp"的 manpage 是放在"/usr/man/man1"底下，"kill","link"是放在 "/usr/man/man2"底下。但如果你自己寫了一個程式而且也寫了一個 manpage 當使用指南，方便自己或別人做線上查詢，這個非標準的 *UNIX* 線上查詢資料要放在哪裡呢？答案是：隨便你！

比如說，該 manpage (troff/nroff format) 叫"my_program.1"，放在你的"~/man/man1"（注意，".?" 的就放 "man/man?"底下），那你只要把你的 MANPATH 這個環境參數重新定義成：

```
% setenv  MANPATH /usr/man:$HOME/man/man1
```

就可以了[7]。man 這個指令就是依照 MANPATH 來找 manpage 的！

LD_LIBRARY_PATH 各位寫 C program 都知道，大部分的 program 都重複 call 到同樣的 function，用到的 library 也一樣，因此，如果每個程式的執行檔都"含"有那些 library 的話就很浪費，所以 *UNIX* 是只把你要用的 library 名字放進程式內，然後在該程式被執行時，才把要用到的 library link 進執行檔再執行。這個觀念叫 "Shared Library"。LD_LIBRARY_PATH 是用來定義你執行程式時，系統搜尋 library 的路徑順序。所以有時候你執行某個 program 會得到像

```
ld.so: warning: /usr/lib/libc.so.1.6 has older revision than expected 8
```

的 message 就是因爲在 link 過程有問題(版本不對)的緣故。一般，如果只是 warning，則可以不理會，程式仍可以正常運作，否則，若是出現類似

```
ld.so: libc.so.101: not found
```

的訊息，那可能要重新 compile 程式了。

EDITOR 這個環境參數用來設定你的編輯器。系統的預設值是"vi"。你可以重新設定，如：

```
% setenv EDITOR /usr/openwin/bin/textedit
```

XAPPLRESDIR 這個環境參數用來設定到那裡去找 X Window 程式的"resource"，如：

```
% setenv XAPPLRESDIR /usr/lib/X11/app-defaults:/users/X11/app-defaults
```

NNTPSERVER 跟網路有關，用來定義 news server ，請自行參考相關文件。

好了，簡單說到這裡，應該把大家在 C Shell 裡面會用到的一些技巧都說過了。其實也沒有很多，但只要大家活用，你會發現其組合有千百種，要知道更詳細和更高竿的用法，可以參考在最後一章所列的參考書籍。

在下一章，我們要用兩個例子來解釋".cshrc"及".login"，可以看看如何將前面談過的觀念用在這兩個檔案內。

[7] 寫進".cshrc"或在殼裡面下這個指令。

科學家因積習，或老天的安排，
往往有所偏執。

from Chaos
by James Gleick

Chapter 5

看看幾個實際的例子

在這一章，我們用實際的兩個啓始檔案：".cshrc"和".login"來總結前兩章的說明。另外，透過逐行逐字的解說，也將前面兩章所遺漏的內容加以補足。最後再介紹另一個簡單的"結束"檔案：".logout"。先來看".cshrc"檔。注意，最前面的號碼是爲了說明方便加上去的，不在原來的檔案中。

5.1 One Example - .cshrc

```
 1 ############################################################
 2 #   Other settings of 'setenv' are all in ~/.login         #
 3 ############################################################
 4 setenv OPENWINHOME /usr/openwin #define OpenWindows' Home directory
 5 setenv XAPPLRESDIR '/usr/lib/X11/app-defaults:/usr/local/X11R5/lib/X11/app-defaults'
 6 setenv SHELL  /bin/csh
 7
 8 # Everytime the files listed in the following got changed, you will be notified.
 9 # By default, your mailbox, the SA' memo and bulletin board are the ones you
10 # should be informed
11
12 set mail = ( /usr/spool/mail/`whoami` /usr/local/lib/adm/messages )
13 umask 002
14 set nobeep
15 set nonomatch # man csh to learn more details about 'nonomatch'
16 # set the filename self-explaining feature. i.e Enable filename completion.
17 set filec
18
19 set path = (   $OPENWINHOME/bin $OPENWINHOME/bin/xview $OPENWINHOME/lib \
20              $OPENWINHOME/lib/X11 \
21              $OPUS_HOME/bin \
22              /usr/local/X11R5/bin \
23              /bin /usr/bin /usr/ucb /usr/5bin /etc \
24         /usr/etc /usr/games /usr/local/bin ~/bin  . )
25 set path = ($path /usr/tran/sparc/bin $OPENWINHOME/demo)
26 ############################################################################
```

```
27 #set noclobber  # Redirect output protection
28 # set prompt="`hostname`[`pwd`]\!> "
29
30 set prompt="`hostname`/\!> "
31 set history=40
32 set savehist=40
33 set ignoreeof
34 set time=100
35
36 alias a alias
37 a bye exit
38 a ls 'ls -F'
39 a rm 'rm -i'
40 a mv 'mv -i'
41 a cp 'cp -i'
42 a vt100 'setenv TERM vt100'
43 a moer more
44 date
45 pwd
46
47 a h history
48 a md mkdir
49 #a cd 'cd \!* ; pwd'
50
51
52
53 # a cd 'cd \!* ; set prompt="`hostname`[`pwd`]\\!> "'
54 a lo logout
55 a mail /usr/ucb/mail
56 #a cprv 'hz2ps -big -hf hku-ch 10 1 -hbm 16 -ls 12 -v \!* |lpr'
57 #a cpr 'hz2ps -big -hf hku-ch 10 1 -hbm 16 -ls 12 \!* |lpr'
58
59 a tin "setenv EDITOR celvis; /usr/local/bin/tin"
60 a dir "ls -laF \!* | more"
61 a elm "setenv EDITOR vi;/usr/local/bin/elm"
62 a ll 'ls -la \!* | more'
63 a df '/bin/df | more'
64 ###########################################################################
65 if (`uname` == "HP-UX") then
66     set console_user = `w | grep console | grep -v grep | cut -f1 -d' ' '`
67     echo Local user is : $console_user
68     if ( $console_user != `whoami` ) then
69         echo "Remote login ... not on console"
70         #echo Machine Type is HP
71         #echo setting up proper stty modes
72         stty intr ^C
73         stty erase "^?"
74         stty susp ^Z/^Y
75         stty kill ^U
76         #stty werase ^W  # By defaults, no need to set again
77         stty stop ^S/^Q
78         stty eof ^D
79     else
80         echo Login on Locally ...
```

```
 81     endif
 82
 83     unset autologout
 84     alias a alias
 85     a df 'bdf | more'
 86     a psf 'ps -ef'
 87     a psl 'ps -el'
 88     a rsh remsh
 89     a pstat  swapinfo -m
 90
 91     set path = (. /etc /usr/etc /bin /usr/bin /bin/posix \
 92             /usr/contrib/bin  /usr/bin/X11 \
 93              /usr/vue/bin /usr/local/bin /usr/local/X11R5/bin ~/bin )
 94
 95 else # uname == SunOS
 96
 97     limit coredumpsize 5m
 98
 99 endif
100 ##########################################################################
101 # Do I have my personal command aliases and other SHELL settings  ?
102 # You can change the name anyway, e.g My_Own_Cshrc --> ~/My-My
103 ##########################################################################
104
105 if ( -e ~/My_Own_Cshrc ) then
106 #   echo "Found my own setting files  ... sourcing....."
107     source ~/My_Own_Cshrc # source ~/My-My ? if you want ..
108 endif
109 ##########################################################################
```

這個例子一開始是以"#"號爲首的一段說明，在 C Shell 裡，以"#"號爲首的行，都被視爲"註解"，不列入處理。

第 4、5 行分別定義了"OPENWINHOME"和"XAPPLRESDIR"這兩個環境參數，注意在第 5 行中，當我們要定義的環境參數需要兩個以上的路徑名稱時，要用"："分開這些路徑。

第 6 行定義"SHELL"這個環境參數。這個環境參數是告訴系統你用的殼是哪個，有些程式，例如，mail, vi, ex 等，允許你從中暫時跳出來執行其它程式，它們會先看"SHELL"這個環境參數，再來決定要用那種殼來執行其它程式。這個例子是設定爲"/bin/csh"，就是 C Shell。

第 7 行是空白行，跟以"#"號爲首的行一樣，都是不被處理的，你愛空幾行就幾行，另外，該有空白的地方，只要是一個或一個以上的空白都可以。

再看第 12 行，你應該知道意思吧[1] ？

第 13 行要特別說明。你有沒想過，當你產生一個檔案時，系統會把它的權限設定成什麼？對一般檔案而言，它的預設值是：666，也就是所有的人都能讀、寫該檔案。如果是目錄，它的預設值是：777，也就是所有的人都能讀、寫、搜尋、進入該目錄。

前面提到，你可以用"chmod"指令來改變一個檔案的權限屬性，現在你看到的第 13 行是告訴系統，對於新產生的檔案要如何定權限，不要是 666，777。這個設定叫做"umask"（user mask 的縮寫）。這個變數，顧名思義，就是告訴 C Shell，哪些權限要關掉（mask，蓋住）。以這個

[1] 你沒看第四章嗎？

例子來說，"umask 002"，"002"的表示法跟以前講"chmod"一樣，是以三段式來分別設定使用者本人、同 group 的人、其它人的關閉權限。所以"002"是表示：使用者本人及同 group 的人都不關閉權限，但對其它人的權限應該關掉"寫"這個權限。記得嗎，"2"以二進位制表示是："010"，這個"1" 在"rwx"中對到"w"（中間那個），就是把"w"給蓋住，不要那個權限，又因爲原來應該是"666"，所以現在變成"664"了：

```
原來 ： 6      6      6   <--->   110    110    110
umask： 0      0      2   <--->   000    000    010
變成 ： 6      6      4   <--->   110    110    100
```

也就是說，所有新產生的檔案，它們的檔案權限都會變成"664"，而非"666"；所有新產生的目錄，它們的檔案權限都會變成"775"，而非"777"。你可以試試改變"umask"來觀察新建檔案、目錄的權限，並決定哪個數字才是你所需要的。

第 14 行是告訴系統不要嗶嗶叫。比如說在使用 file completion 功能時，如果 C Shell 沒找到可以 match 的檔案名字，它會叫一聲。你可以用"set nobeep"叫它閉嘴！另外在用"vi"時，按了太多的<ESC>鍵雖然無害，但它也會嗶嗶叫，嫌它煩，可以用"set nobeep"請它不要叫。

第 15 行是告訴系統，在沒找到可以 match 的檔案名字時，不要說"no match"。下面是個例子：

```
% set nonomatch
% ls Z????????
Z???????? not found
% unset nonomatch
% ls Z????????
No match.
```

看出不同了嗎？

第 17 行應該不必解釋吧？

第 19 行是設定"path"。注意，在這裡用到"$OPENWINHOME"來代替之前定義過的"/usr/openwin"。在這一行的尾巴用"\"來表示這個指令還沒有結束，下行待續的意思。因爲要設的"path"太多了，所以分行寫。一直到第 24 行，最後一行了，不必再"待續"，所以沒有"\"在行尾。

要特別提醒大家，如果在 path 中將"."包含進去，可能會惹來大麻煩，怎麼說呢？比如，你現在"cd /tmp"，然後想看看在那裡有些什麼檔案，就下指令"ls"。這是很理所當然的反應，但你可能沒想到，在"/tmp"下有個檔案型態是"executable"的檔案就叫做"ls"，內容是"\rm -rf ～"！你執行了"ls"，結果卻把自己所有檔案全給殺了！原因只在，你將"."放在 path 的第一個，當你下指令"ls"時，系統先在"目前"目錄找"ls"，眞的找到了，就做"\rm -rf ～"的動作了。爲了安全起見，還是將"."從 path 中拿掉吧，但是這樣一來，你就得用"./run"來指定執行目前路徑下的程式"run"，而不能只是下指令"run"了，這是比較不方便的地方。

第 25 行，意猶未盡，"path"還不夠，追加一段。注意，在這裡用"$path"來表示剛剛定義過的"path"（19-24行）。

第 30 行是設定系統提示號。先看‘hostname‘這個東西。在 C Shell 中，我們用 ‘ 符號來取得某個指令的輸出結果。所以"set ABC = ‘hostname‘"的意思是把"ABC"這個變數的值設爲機器的名字：系統先執行等號右邊的"‘hostname‘"，再把結果放到"ABC"。下面這個例子可以清楚顯示利用"‘"的作用：

```
% hostname （系統執行 hostname 指令）
```

```
roxy
% `hostname`    (系統執行的是 roxy 指令)
roxy: Command not found.
```

在`hostname`之後的!是用來取得歷史編號，每當你下完一個指令，這個"!"所代表的值就會加一。那爲什麼在"!"前面還要加上一個倒斜線呢，因爲如果只寫"!>"，系統會把">"當成歷史編號或事件，在歷史清單中找，而我們的意思是要">"這個符號，爲了叫它不要把"!>"當成是找歷史，就在前面用倒斜線來把驚嘆號先包住。經過第 30 行這個指令的定義後，系統提示號會變成：

```
% set prompt="`hostname`/\!> "
roxy/23>□
```

第 31 行應該不必說了。

以前說的歷史清單，都只針對一次登錄（login）而言，就是說，這些歷史清單在你離開 *UNIX*，下次再登錄進來時，就都不見了。如果你想把這次的歷史留到下次也能用，就利用第 32 行這個變數"savehist"，它告訴系統，把這次用的指令存起來，放到"~/.history"中，這樣，下次你 login 時，系統會去這個檔案中把上次的歷史再抓回來當成已經有的歷史。

第 33 行請你自己解釋吧！

你可能碰過，在執行完某個指令後，系統會跑出一段數字像這樣：

```
% cat *.ps *.log
1.1u 0.3s 0:15 10%
```

又爲什麼有時候有這些訊息出現，而有時候卻又沒有呢？

第 34 行的"time"變數，用來規定，當指令執行時間超過多少"秒"的時候，才顯示這些東西，以上例而言，這些數字代表的是：該指令共用了 1.1 秒的"user time"；0.3 秒的"system time"；15 秒的"real time"；其中前兩項總和佔第三項的百分之十。要注意的是，這個時間其實並不可靠，僅供參考而已。第 34 行的意思是，當指令執行時間超過 100 秒時，才顯示這些系統資源使用上的資訊，少於 100 秒的話，就省了。

第 36 行定義一個叫"a"的別名，它實際上是"alias"的意思。

第 37 到 43 行，你應該可以解釋吧。

記得前面說過，在你一 login 到 *UNIX* 時，系統會先去執行你的"~/.cshrc"，所以在第 44 行和 45 行的指令，只是告訴系統，在我 login 之後，要將目前所在目錄路徑和現在的日期、時刻顯示出來。

第 59 行是在一個"alias"中，先後做兩件事：先做"EDITOR"的定義，再執行程式。之間用"；"分隔兩個指令。所以當你下"tin"這個指令時，系統先"setenv EDITOR celvis"，然後再執行"/usr/local/bin/tin"。

第 60 行的倒斜線跟前面說的定義"prompt"時一樣。這一行是定義一個叫"dir"的別名，當你下"dir"指令時，它執行的動作其實是："ls -laF !* | more"。第 62 也是。這個"!*"是什麼意思呢？就是"前一指令的所有參數"的意思[2]。但是就這個 alias 動作而言，當你說：

```
% dir file* unix.txt
```

[2] 參考第四章，講 C Shell history 那一段。

的時候，"file* unix.txt"就算是"前一指令的所有參數"了，然後，實際上執行的是：

```
% ls -laF file* unix.txt | more
```

第 65 行到 99 行，是一個先做條件判斷的小 C Shell 程式。首先利用"uname"這個指令來判斷你所使用的系統是"HP-UX"或者是"SunOS"；兩種機器的使用有些許不同，分別做不同的設定。要注意的是，這個例子並不一定適用在讀者的使用環境，讀者應該根據自己的需要寫自己的程式。特別注意第 97 行的"limit coredumpsize 5m"，它是告訴 C Shell[3]，如果有 coredump 發生時，將 core 的大小限制在 5 Mega Byte，不要超過。這是希望節省磁碟空間，有關 core 的說明，請看本章最後一節。

第 105 行到 108 行是另一小段程式，先看看是不是有另一個設定的檔案，需要在啓動時讀進來做其它設定。這裡的"～/My_Own_Cshrc"你可以把它視爲是"～/.cshrc"的一部份，把它另外放在一個獨立的檔案只是方便管理罷了。把"～/My_Own_Cshrc"的內容整個放進來，也是一樣的。

特別注意"source"這個指令，它是用來"執行"一個 C Shell Script 的。在第一章就說過，當你一進入 *UNIX* 的時候，系統根據你所用的殼，設定你的環境，"看" 的就是".cshrc"這個檔案，怎麼"看"呢，就是"執行"".cshrc"，這裡說的"執行"就是"source"這個指令。這個指令常在你更改".cshrc"、".login"等檔案後，用來強迫系統去認識新的設定。例如，你在你的".cshrc"新增了一個別名"alias m more"，你在".cshrc"中加了這一行後，系統並不知道你有設定這個別名，因爲它只在你新進一個殼的時候才會去看你的".cshrc"，現在你偷偷加了東西，它根本不曉得，要讓它知道，就是用："source ～/.cshrc"。它會逐條執行你的".cshrc"，做完後又等著你給指令。所以，你就可以用這個指令在不同的應用中設定不同的環境，像在上面這個例子中，就是先判斷你有沒有"～/My_Own_Cshrc"這個檔案，如果有的話，就"執行"它[4]。

5.2 One Example - .login

下面要看的是".login"這個檔案的一個範例，主要是給 Sun 工作站使用的，在使用者 login 之後，自動根據使用者的選擇，進入視窗；在使用者跳出視窗之後，自動 logout。各位參考看看，有看不懂的地方，先請大家就近請教你的系統管理員或者問問其他可能可以給你解答的人。或者在看過第七章之後再回頭來研究。這裡要說明的只是，爲什麼把"setenv"的東西都（大部分）放在".login"而不是".cshrc"？

這是因爲，"setenv"的環境參數只須設定一次，然後它們的值就會傳到以後的每個後續的殼裡面[5]，所以把它們放在".login"中，只要做一次就夠了；當然，把這些"setenv ..."放在".cshrc"也是可以的，只是，這樣一來，你每進入一個殼，這些"setenv"的環境參數就又被重複設定一次，有點多此一舉罷了。

```
 1 setenv EXINIT 'set sh=/bin/csh sw=4 ai report=2' # Set default EX/VI mode
 2 setenv NNTPSERVER 140.96.200.1      # This is for news reader 'tin'
 3
 4 ################ This line sets for X's resources files    ###############
 5 setenv XAPPLRESDIR '/usr/lib/X11/app-defaults:/usr/local/X11R5/lib/X11/app-defaults'
 6 ################ This line sets for X's resources files    ###############
 7
 8 ################ This line sets your man-pages search path   #############
 9 setenv MANPATH ${OPENWINHOME}/man:/usr/man:/usr/local/man:/usr/local/X11R5/man
```

[3] 有些 unix 系統的 C Shell 並沒有這個參數可設。

[4] 最後兩段程式是簡單的 C Shell Programming，語法和 C 語言很像，在第七章筆者會教大家一些簡單的程式寫法。如果上面這兩小段程式各位看不懂，等各位讀過第七章後，再回過頭來看，應該可以看得懂的。

[5] 請參考第四章的講解。

```
10 ################ This line sets your man-pages search path   #############
11
12 ########## This line sets your shared libraries search path   ############
13 setenv LD_LIBRARY_PATH .:/usr/local/X11R5/lib:${OPENWINHOME}/lib:/usr/lib:/usr/lib/X11
14 ########## This line sets your shared libraries search path   ############
15
16 ##########################################################################
17 #   Printer setting                          #
18 ##########################################################################
19 setenv PSLIBDIR /usr/tran/sparc/lib
20 setenv TROFF ptroff
21 setenv TCAT "lpr"
22 setenv PRINTER lp
23
24 ################################################################
25 #   The following sets for Chinese X-Terminal: cxterm       #
26 ################################################################
27 setenv CXTERM_FONTPATH '/usr/local/lib/cxterm_fonts'
28 setenv FONTPATH "$OPENWINHOME/lib/fonts:$CXTERM_FONTPATH"
29 setenv HZINPUTDIR '/usr/local/lib/cxterm_dict/citnf' # cxterm_NewFace
30 ##############################################
31 #   End of setenv's                          #
32 ##############################################
33
34 ##########################################################################
35
36 if ($TERM != "sun") then
37 eval `tset -sQ -m dialup:?vt100 -m switch:?vt100 -m dumb:?vt100 $TERM`
38 endif
39
40 # If possible, start the windows system.  Give user a chance to bail out
41
42 #=================================
43
44 set myname = `hostname`
45 setenv XSERVER $myname
46 setenv XCLIENT $myname
47
48 #=================================
49 # Automatically 'setenv TERM'
50 set noglob
51 eval `tset -sQ`
52 set glob
53 if ($TERM == "network" || $TERM == "dialup") then
54
55    setenv TERM vt100
56    unsetenv TERMCAP
57
58 endif
59
60 echo "Terminal type is $TERM"
61 #=================================
62
63 echo ""
```

```
 64 if ( `tty` == "/dev/console" && $TERM == "sun" ) then
 65     click -n    # click -n turns off key click
 66 endif
 67
 68 cd
 69 echo ""
 70
 71 if ( `tty` != "/dev/console" || $TERM != "sun" ) then
 72     set prompt="`hostname`/\!> "
 73     exit    # leave user at regular C shell prompt
 74 endif
 75
 76
 77 set mychoice=op
 78 ###############################################################################
 79 switch( $mychoice )
 80 case    op:
 81     unset mychoice
 82     echo  "starting X-Windows .... "
 83     echo " "
 84     echo " "
 85     echo " "
 86     echo "Which Window Manager ?"
 87     echo "Choose 1: for Open Look Virtual Window Manager (olvwm)"
 88     echo "Choose 2: for Motif Window Manager (mwm)"
 89     echo "Choose 3: for Open Look Window Manager (olwm)"
 90     echo -n "Choice: (1,2,3) "
 91
 92
 93     set window_manager = $<
 94     switch ($window_manager)
 95     case 3:
 96
 97          # Use Open Look Window Manager
 98         /bin/sed "s;olwm;olwm -3;" ~/xinitrc.template.no_kill > ~/.xinitrc
 99          clear
100          echo " "
101          echo " "
102          echo " "
103          echo " "
104          echo "Starting Open Look Window Manager, please wait ..."
105          echo " "
106          echo " "
107          echo " "
108          /usr/5bin/banner " O p e n"
109          echo " "
110          echo " "
111          /usr/5bin/banner " L o o k"
112          breaksw
113     case 1:
114
115          # Use Open Look Virtual Window Manager
116          /bin/sed "s;olwm;olvwm;" ~/xinitrc.template.no_kill > ~/.xinitrc
117          clear
```

```
118          echo " "
119          echo " "
120          echo " "
121          echo " "
122          echo "Starting Open Look Virtual Window Manager, please wait ..."
123          echo " "
124          echo " "
125          echo " "
126          /usr/5bin/banner " O p e n"
127          echo " "
128          /usr/5bin/banner " L O O K"
129          echo ""
130          /usr/5bin/banner " Virtual"
131          breaksw
132
133
134     default:
135          /bin/sed "s;olwm;mwm;" ~/xinitrc.template.no_kill > ~/.xinitrc
136          clear
137          echo " "
138          echo " "
139          echo " "
140          echo "Starting Motif Window Manager, please wait ..."
141          echo " "
142          echo " "
143          echo " "
144          echo " "
145          echo " "
146
147          /usr/5bin/banner "M o t i f "
148          breaksw
149     endsw
150
151     #sleep 3
152
153     op
154     clear_colormap  # get rid of annoying colourmap bug
155     clear       # get rid of annoying cursor rectangle
156     logout      # logout after leaving windows system
157     breaksw
158     #
159
160
161 case    sunview:
162 default:
163     unset mychoice
164     echo -n "starting SunView (Control-C to interrupt)"
165     sleep 3
166     # default sunview background looks best with pastels
167     sunview  -b 200 255 200 -background $HOME/cartoon.sun
168     clear       # get rid of annoying cursor rectangle
169     # logout        # logout after leaving windows system
170     breaksw
171     #
```

```
172
173 endsw
```

5.3 One Example - .logout

在這之前，筆者一直沒提到這個檔案，因爲它不是頂重要。".logout"這個檔案，跟".login"剛好相反，它是在你跳出 *UNIX* 時，告訴系統在你離開之前所要做的事。以下是一個很簡單的 ".logout" 例子:

```
# This is .logout file
clear
echo " "
echo " "

echo -n "`whoami` logging out at : "
date | tee ~/.lastlog
echo " "
echo " "
echo " "
```

你可以試著解釋看看，或者把它鍵入你的"～/.logout"，看看在進出*UNIX* 幾次後，"～/.lastlog"有什麼變化。

5.4 有些事情你最好先知道

到目前爲止，各位可以發現，*UNIX* 給予使用者非常大的自由度，也因此，使用者常常被它驚人的能力所吸引，但是 *UNIX* 實在是太高深了，使用者常常在不知不覺中就會"誤觸地雷"，尤其是那些在電腦反應稍慢時，就喜歡亂敲鍵盤的人來說， *UNIX* 就常常回報以讓新手手足無措的結果。

✍ 有個很大的檔案叫 core，它是怎麼跑出來的？

core 這個檔案，是自己跑出來的，當某個系統或程式當掉時，就會有 core (core dump)。這個檔案記錄了該程式當掉當時系統的所有狀態（在記憶體內），之所以要做這個記錄，是爲了 debug 程式用的。所以如果你是跑你自己的 C 程式導致 core dump ，你可能會用得到它：在 debug 時，例如 Sun 上面的 dbxtool 就會自動去參考 core 這個檔，自動將指標停在當掉的那一行程式，以便你快速的找到 bug。如果你不需要 debug 程式，建議你殺掉它，免得佔磁碟空間。至於 core 這個名字的由來，也是有歷史淵源的。在早期，電腦的記憶體並不是現在的 IC，而是一個個的磁圈 (magnetic core)，利用其極性來代表 0 與 1，當時爲了偵錯，在系統當掉時，就把磁圈的資料記錄下來，慢慢找出錯誤，後來，科技進步了，記憶體都是用半導體 IC，不再用磁圈，但"core"這個名字還是沿用下來了。你可以用下面這個指令來找尋你的目錄下有無 core，常常做這個動作是被鼓勵的，這樣可以避免不必要的磁碟空間浪費。

```
% find ~ -name core -print
```

✍ 奇怪的檔案名字。

常常，你會發現你的目錄下面有幾個奇怪的檔案名字，你沒有刻意產生它，它就在你不注意的時候，或者由於程式的錯誤，或者由於一陣亂敲鍵盤之後，反正它就在那裡，殺也殺不掉。這種檔案，根據經驗，以"-"開頭的最多，例如："-rwx--x--x"。檔案名字以"-"開頭的殺法其實在"rm"這個指令的 manpage 裡面都會特別提到，大概是大家不習慣用"man"，它老是被忽略，以至於讓它變成一個超級 FAQ[6]。在 SunOS 裡，它甚至是第一個講到的 rm 選項：

```
OPTIONS
     -    Treat the following arguments as filenames '-' so  that
          you can specify filenames starting with a minus.
```

只是大家不注意罷了。

```
% rm - -rwx--x--x
```

當然還有其他方法，例如：

```
% rm ./-rwx--x--x
```

若是其他奇怪的名字，還有更通用的方法：

```
% rm -i *
```

注意要用"-i"選項，讓系統一一問你要不要殺，碰到你要殺的那個檔案時回答"y"，其他都回答"n"，或者用"Control-C"中止"rm"的動作。如果連這一招也不行，只好用"i-node number"來殺了：

```
% ls -i *                                用 -i 選項先找出檔案的 i-node 號碼
% find . -inum 1234 -ok rm '{}' \;              殺掉該檔
   或
% find . -inum 1234 -ok mv '{}' normal_name \;  或者改名
```

1234 是你要殺的那個檔案的 i-node 號碼。i-node 是 *UNIX* 眞正用來區別檔案的方法，就像我們每個人的身份證號碼一樣，由於是更低階地觸及 *UNIX* 系統，自然威力就更大嘍!

特別提醒你，當你要殺掉檔案名稱以"."開頭的檔案時，使用下面這個指令，在某些 *UNIX* 系統會造成難以彌補的災難:

```
% \rm -rf ./.*
```

爲甚麼呢? 注意上述指令的"*"使得"./.*"把"./."和"./.."也包括了，又有"-rf"選項，表示沒有反悔的餘地，目錄也不放過，其結果是連目前所在的這個目錄也全給殺掉了! 使用這兩個選項千萬要小心!!

另外，如果是較普通的特殊字元，例如"&"，"|" 等，則可以用倒斜線"\"來對付，下面的例子是將一個名爲"a&"的檔改名爲"strange.file" :

```
% mv a\& strange.file
```

[6] Frequently Asked Question 的縮寫。

✍ 一行有太多字了，如何把每一行的字"折"到下一行？

用"fold"這個指令。例如：

```
% fold -80 input > output
```

是把 input 這個檔案的每行字，超過 80 的部分折到下一行。

✍ 如何把 man 指令的結果存成普通的 Text 檔？

第一個想法一定是：

```
% man 指令名 > ascii-file
```

這樣只成功一半，因爲這樣產生的檔案裡頭，會含有大量的控制字元，用"more man-file"看它時還看不出來，但你用"vi man-file"或者將它從印表機列印出來時就可以發現。當你並不需要這些控制字元時，應該要濾掉它們，用：

```
% man 指令名 | col -b > ascii-file
```

這些控制字元靠"col -b"指令被過濾掉了！另外，如果你要直接由印表機印出 man 的結果，可以用：

```
% man -t 指令名
```

如果你拿到的是所謂的"man file"或"manpage file"，以 roff 寫的，可以用

```
% nroff -man file.man | more
```

來看。或是：

```
% nroff -man file.man > file
```

存到一個檔案去，再做處理。

✍ 如何把 DOS 格式的文字檔轉成 *UNIX* 的格式？

如果你在 *UNIX* 上看到一些文字檔，它們每一行的最後都有個"^M"，那種檔案大概原本是在 DOS 上的，你可以用下面的方法將它轉成 *UNIX* 上的文字檔:

```
% cat dos_file | col -b > unix_file
```

有些 *UNIX* 系統，如 SunOS，又有一個叫"dos2unix"的指令[7] 用來做這件事：

```
% dos2unix dos_file > unix_file
```

✍ *UNIX* 有 DOSKEY 的功能嗎？

會問這個問題的人，十之八九是 DOS 的使用者，用慣了以箭頭鍵來叫用過去下過的指令，一到了*UNIX* 就手忙腳亂，按了箭頭，卻什麼也沒發生，甚至出現亂碼。

這個問題其實應該換成"哪種殼有提供類似 DOSKEY 的功能"，因爲，這種功能應該由殼來完成。所以，如果你眞的要這個功能，換個殼吧！像"tcsh"、"bash"等都有這個功能。當然還有其它的，你只要拿得到就可以換殼，這也是 *UNIX* 可愛的地方之一。

[7] 對應的指令是"unix2dos"。

初學者常常抱怨 *UNIX* 的指令名字令人一頭霧水，不知是哪裡來的。在一開始，我們已經看過一些常用指令，發現他們的名字幾乎都是縮寫來的，很好記，但仍有一些 *UNIX* 指令，他們的名稱就有得研究了。下面我們以輕鬆的心情來考考古，看看它們的由來。

awk　awk 這個指令一般的初學者並不會用到，但它實在是一個功能強大的"程式"。這個程式的作者是三個在 *UNIX* 史上赫赫有名的人物：Alfred Aho, Peter Weinberger, Brian Kernighan。Alfred Aho另外還是 egrep 這個程式的作者；Peter Weinberger 還寫了 lcomp； Brian Kernighan 就是頂頂有名的 C 語言發明人之一。C 語言聖經—"The C Programming Language"中，人稱"K&R"的"K"就是指他。[8] 他另外還寫了 ditroff 、eqn 等程式。

還沒講到正題：awk 名字就是以這三個人的姓來命名的。

biff　這個程式用來通知使用者，有新的電子郵件送達，並立刻顯示其中的幾行在螢幕上讓使用者參考。可是，爲什麼叫"biff"呢？根據筆者親自[9] 向 Heidi Stettner 求證，她說，當時她是柏克萊的研究生，養了一隻小狗，每天跟她到學校去，很受大家的喜愛，都把牠當成幸運物，而這隻狗喜歡在郵差送信來的時候吠，後來當 John K. Foderaro[10] 完成這個程式時，想不出更好的名字，就眞的把這隻狗的名字"biff"用上了！這隻幸運的狗跟隨 Heidi Stettner 小姐過了一生，已經在 1993 年 8 月去世，享年 15 歲，也算不虛此行了！

另外，根據筆者看過的一本書上寫說，"biff"甚至還和主人一起上過一門"compiler"的課，而且拿了個"B"，在學生名單公佈欄上甚至出現牠的照片和名字，底下還寫著：攻讀"Ph. Dog"中。

rc　你一定覺得奇怪，爲什麼用來規劃 C Shell 的啓始檔案叫做".cshrc"，"rc"到底是什麼意思？ rc 的原意是"RunCommand"。早在 1965 年，MIT 的 CTSS 系統就有個"runcom"機制，用來在一個檔案內執行一串指令，類似 batch file 的觀念，當時叫"runcom"，取"run commands"的縮寫。後來，到了 *UNIX* ，凡是這種在一個檔案內執行多個動作的啓始檔案都以"rc"結尾，例如".cshrc"，或者，在系統開機時的啓始檔"/etc/rc"等等。

C++　C 語言後來又演進到了 C++，由當時在貝爾實驗室的 Bjarne Stroustrup 所設計。之所以取名 C++，是因爲在 C 語言中有個極特別的語法：在變數後面加上"++"，表示變數值增一。 C++ 的名字就是這樣來的。

rsh　不，不是要講它是"remote Shell"的縮寫。常常，我們會要到別架機器上去暫時執行某個程式、指令，而用 rsh。例如"rsh ccsun2 program-name"。但是如果是要丟背景程式，就不是直接用：

```
% rsh ccsun2 program-name &
            或
% rsh ccsun2 "program-name &"
```

這樣簡單了，因爲 rsh 會等到該程式結束後才會在回到本機台的Shell。解決的方法是：

```
% rsh ccsun2 -n "program-name >& /dev/null < /dev/null &"
```

除了".cshrc"，".login"是要注意的外，另一個叫".rhosts"的檔案也是需要知道的。前面說過，用"rlogin"、"rsh"時可以不必再給 password 就能登錄到別的機器，如果你的 SA 沒有設定的話，你也可以自己設定你要"信任"的機器。先看個例子，如果你的"~/.rhosts"是：

[8] "R"是指 Dennis Richie, unix 的創始人之一。

[9] 有人質疑有關 biff 的這些資料是筆者沿用網路上流通的 unix FAQ 來的。在這裡要特別聲明，筆者確實曾以 Email 向 Heidi Stettner 確認有關 biff 故事的正確性。因未得同意，無法將往來的 Email 公開在此。

[10] 根據 Heidi Stettner 的說法， Bill Joy 應該也有參與。又，John K. Foderaro 也是 Franz Lisp 的作者。

```
ccsun1
roxy
metaford
robbins
```

表示這幾架機器是你信任的機器，當這幾架機器有"rsh"或"rlogin"的要求連線時，你都允許。特別注意，不要把不相干的機器放進來，因爲，如果該機器正好有個使用者的 username 和你一樣，那就等於允許他自由進出你的系統了，一旦他用"rsh" ，"rlogin"連線，系統會把他當成是你！千萬注意！

Time is linear
Memory is a stranger
History's for fools
Man is a tool in the hands
Of the great God Almighty
And they gave him command
Of a nuclear submarine
And sent him back in search of
The Garden of Eden

from Amused To Death
by Roger Waters

Chapter 6

Tour of Tools

這一章是在前一版（Ed. 2.4）來不及寫進來的，很多有用的 *UNIX* 程式都因爲功能太多或操作太複雜，讓初學者望而卻步，平白錯失很多好用的工具。這一章要講的，主要是讓這幾個實在好用的工具能很容易被初學者接受，並用來處理日常的工作。

現在就開始吧...。

6.1 V-eye

"To vi or not to vi, that is a question!"。這是實在話，筆者剛開始用 *UNIX* 的時候也曾讓 vi 這個"惡名昭彰"的編輯器教訓了一陣子。

但是，我要收回前面說過的話，不會用"vi"的，現在請你好好看完這一章，試著用用看[1]。

"vi"是"visual editor"的縮寫。這是個令人又愛又恨的編輯器（Editor），我想沒有一個 *UNIX* 初學者會喜歡它(Well，如果你是一個 *UNIX* 初學者，而你馬上就喜歡用"vi"，那你眞的跟別人不一樣，非常不一樣!)。爲什麼要學它呢？原因很簡單："vi"是 *UNIX* 裡面既定的（default）編輯器，只要是 *UNIX* 系統，就一定會有"vi"這個編輯器，什麼版本 *UNIX* 都一樣。學會以後，走遍天下的 *UNIX*，你都不怕沒編輯器可用，而且，你還可以大聲跟別人說：我會用"vi"！

以下介紹的只是"vi"很小一部分的功能，但是你只要會這些就夠了，絕對綽綽有餘。再說一次，眞的很少，不要怕，也不要讓所有有關"vi"的壞話影響你。

6.1.1 小心你的鍵盤

有人說，電腦鍵盤上前三名最容易耗損的鍵是：用來更正打字錯誤的 Back Space 鍵、完成輸入動作的 Return 鍵，以及空白鍵 Space Bar。那是因爲這三個鍵的使用率特別高的緣故。不過，對一個"vi"的使用者而言，ESC 鍵可能要擠進這前三名。

討厭"vi"的第一個理由是它的"雙模式"。什麼叫"雙模式"呢？就是在輸入資料和處理資料的時

[1] 如果眞的學不會，也無關緊要。

候是處在兩個不同的模式中，在切換到另一個模式之前，你所敲進去的字母，會被當成不一樣的解釋。例如，你敲進"a"這個字母，如果你是在"輸入"模式，那麼"a"就會出現在螢幕上；如果你是在"命令"模式下敲進"a"這個字母，那就不會有"a"出現，而是其它反應。

在一般的 DOS 編輯器中，使用者完全處在"輸入"模式，你敲進的每一個字母都是要輸入的文字，而當你要存檔案、或做搜尋、代換等動作時，是用控制鍵來完成，移動游標也只需要直接按箭頭鍵，上下左右直接操作。

"vi"呢？你得先搞清楚你在哪個模式。問題是，誰有那閒工夫，在輸入資料時還記住現在的模式狀態？沒有，初學者更不可能。

所以，你一定要常常按<ESC>這個鍵！沒事兒也去按按，按它做啥？按它可以回到"命令"模式，不管你在哪個模式，按了<ESC>鍵後就一律回到"命令"模式。按幾下都不要緊，一下、兩下、一百下都一樣，回到"命令"模式。

6.1.2 vi 的模式有兩種

當你剛進入"vi"的時候，就是在所謂的"命令模式"，這時候，你可以"處理"檔案的內容。例如，殺掉兩行啦，把"abc"代換成"ABC"啦，"進入輸入模式"啦，等等。反正就是"下令"叫它做工就對了。

6.1.3 眞的來玩玩

下面我們實際用它來編輯一個檔案，看看"vi"到底有多難用。先下指令：

```
% vi
```

這時候，整個殼就被"vi"霸佔了。你會看到最左邊有一排毛毛蟲（"～"），那些毛毛蟲是表示，那些行是沒有資料的，是空的。

我們先練習按<ESC>這個鍵，確定回到"命令"模式，然後才開始一切動作。這是好習慣，練習不看鍵盤就按得到<ESC>，將來你會愛死這個鍵的。

毛毛蟲還在，游標也沒動，停在第一行第一格，這個時候"vi"在等命令。假設我們要輸入一段文章，你要告訴它"我要輸入"。你現在按"i"，告訴它進入"insert"模式，準備接受打進去的字。

你按"i"的時候，是在"命令"模式，這個"i"字母是"命令"，不是你要打的字。在按過一個"i"之後（一個喔，一個就夠了），"vi"就進入"插入"模式，屬於輸入模式之一。後來再打的字都被插入到游標的位置。

好，你按過"i"了，還沒開始輸入資料，螢幕上除了毛毛蟲以外，不該有東西。萬一你已經搞砸了，輸入了些奇怪的東西，罰你按<ESC>十下，然後接著按":q!"。這樣可以跳出"vi"，請你再進來一次。

按了"i"之後，請你輸入這些文字：（不要按"return"鍵）

```
I love UNIX.
```

完成了這段文字輸入，游標還在"*UNIX* ."之後。注意，你還在"插入"模式。現在按<ESC>，隨便

幾下都可以，回到"命令"模式。你發現游標跑到"."上了。這表示，下次再要輸入時，是從"."這個字元開始。

好，按"i"，再進來輸入。打這些字：

```
and I hate vi
```

這時候你發現"*UNIX* "和"and"連在一起"UNIXand"，因爲，你在按"i"再進入輸入模式時，游標是在"."上，表示接著輸入的文字要插在"."之前，所以啦，"and"就貼在"*UNIX* "後面了，而原來在"*UNIX* "後面的"."就被一路推到更後面了。現在，你要改"UNIXand"，將它們折開，也就是在它們中間插入一個空格。

記住，要做非輸入性動作時，先按<ESC>，這時候，游標又跳到"vi"的"i"。

按完<ESC>就是"命令"模式，在這個模式下，你才可以移動游標。如果可以的話，用方向鍵移動游標是再方便不過了，但是在一、二十年前，沒有方向鍵的時候，"vi"還是可以移動游標的，靠的是"H"，"J"，"K"，"L"這四兄弟。這四個鍵一字排開，在整個鍵盤的中間，"H"在最左，所以它用來將游標移左一位；"L"在最右，所以它用來將游標移右一位。"J"則表示下移，"K"是上移。HJKL 四個鍵分別是代表：左下上右，非常好記。

當然，如果你的系統可以用方向鍵來移動游標，就用方向鍵也可以的。

現在要移動游標到"UNIXand"的地方。先按<ESC>，確定已經到了"命令"模式，然後按"H"鍵幾下，再按"L"鍵幾下，讓游標左右跑一跑。先不急著用"J"和"K"，因爲你的資料只有一行，看不出游標上下移動。現在把游標標定在"UNIXand"的"a"上。

到此爲止，仍在"命令"模式，你準備叫它再進"輸入"模式，因爲你要在"a"之前插入一個空格。按下"i"，進入"輸入"模式，再按空白鍵，那麼"UNIXand"就被分開了。再按<ESC>，回到"命令"模式。

好累，對不對？輸入一行字，竟然要跳來跳去，一下子命令模式，一下子輸入模式。這是很多人對"vi"的第一個感覺，也是很多人對"vi"僅有的印象。

先回憶一下剛才做的事：插入資料、移動游標、插入資料。只學了一個"i"命令，就足夠寫完一大段話，當然啦，千萬不要忘了<ESC>！

現在，你先把剛才敲進去的這一段文字存起來，叫"unix.txt"。怎麼存呢？照例先按<ESC>，然後按":w unix.txt"。當你按了冒號（：）之後，在最下面一行會出現冒號，接著你打空格、"unix.txt"，都會出現在冒號之後。"w"是"write"的縮寫，"w unix.txt"表示："把這些內容寫到 unix.txt 這個檔案"。

嚴格說來":"冒號，是"vi"的第三個模式。好吧，我承認剛剛騙了大家，"vi"其實有三個模式：輸入、命令，還有，ed 模式。以冒號（:）爲首的都是 ed 模式。ed 模式主要用來處理檔案內容，例如，搜尋與取代，設定行號等等，等一下我們會看到在 ed 模式下的操作，現在先不管。

存完檔案，現在按":q"（你當然可以先按<ESC>，<ESC>永遠不嫌多），在 ed 模式下按"q"，表示"quit"。到這裡，你跳出"vi"了。

這一趟"vi"之旅，你已經用過"vi"所有的模式，你會：進"vi"、加文字、存檔案、出"vi"。你已經會"vi"了嘛，不是嗎？[2]

剛剛只教了"i"命令，用來輸入。現在教你其它做輸入的命令：

[2] 開始討厭 ESC 鍵了嗎？還是開始喜歡它呢？沒事兒按按它吧！

i	"insert" 的縮寫。在游標之前開始插入文字。
a	"append" 的縮寫。在游標之後開始插入文字。
o	"open" 的縮寫。在游標所在那行的下面，開出一個新行做輸入。
O	"Open" 的縮寫。在游標所在那行的上面，開出一個新行做輸入。
I	"Insert" 的縮寫。在本行最前面（第一格）開始插入文字。
A	"Append" 的縮寫。在本行最後開始插入文字。
J	"Joint"的縮寫。將下一行的內容接到本行。也就是說，去掉本行的跳行字元。

就這樣而已，現在來練習一下：

```
% vi unix.txt
```

跟剛剛不同的是，現在你在"vi"指令的後面加上一個檔案名字，表示是要編輯一個已經存在的檔案，如果該檔案不存在，系統會幫你先造一個空的檔案。

進了"vi"，你看到剛才打的字在裡面，螢幕下方有一行字顯示檔案名、本檔案的行數和字元數。游標照例停在第一個字母。當然，處在"命令"模式。現在按"o"，游標會往下一格，這表示它已經接受了"open a new line"的指示，開了一個新行給你輸入，你可以開始寫了："Learning vi is a good thing." 寫完，照例按<ESC>回到命令模式。再按"o"，游標再往下一格，你又可以再寫一行。當然，正在輸入時按<RETURN>鍵，也是增加一行的意思，跟在命令模式下按"o"同樣意思。

好了，不管你輸入了什麼，對了或錯了，游標在哪裡，都先不管，現在請按<ESC>，然後用"HJKL"或方向鍵移動游標到"good"的"g"字母上。移動游標的要訣是：先往上下（K，J），再往左右（H，L），或先左右、再上下，因爲"H"和"L"在四個鍵的最兩邊，"J"和"K"是連在一起的，盡量不要讓左右和上下混用，增加自己的困擾。

現在按"O"，大寫的"O"。這時，會在游標的上一行多出一行空白，游標停在最前面等著你輸入。輸入："Learning vi is a bad thing."然後按"ESC"回到命令模式。

已經輸入三行了，現在來教大家"改"東西。沒有人能避免打錯字，在"vi"，你不必先把錯字殺掉，再把正確的字填回，你可以直接將正確的字蓋過錯字。

假設你要將"bad"改成"good"。先將游標定在"bad"的"b"字母（當然是在命令模式），然後按"cw"（Change Word），你發現"bad"變成"ba$"，表示"vi"現在要將"bad"完全取代掉，你接著打："good"，"$"就不見了，"bad"也被改成"good"。這時候，還是在輸入模式，你打的東西還會繼續被輸入。 按<ESC>回到命令模式。

OK，如果你堅持要先學會 delete ，現在就教你。

先將游標定在"good"的"g"，按"x"，可以殺掉"g"，就是游標所在的那個字母。再按一次，就再殺一個。如果你要殺掉5個，就按"5x"。這樣夠快吧。

如果你要殺掉整行，那就用"dd"，游標所在的那行就殺掉了。要一次殺 2 行，就用"2dd"，要殺掉一個字用"dw"(delete word)，所謂"一個字"表示有被空白隔開的，例如上面的"I"、"love"、"UNIX"，都是所謂"一個字"。

好了，光是這幾個指令其實就可以完成所有的編輯功能。下面列出幾個你會常用的"命令"：

dd　　殺掉游標所在的那行。

dw　　殺掉游標所在的那個字。

d2w　　殺掉（含游標所在的那個字）以後的2個字。

4dd　　殺掉含游標所在那行的以下 4 行。

dG　　殺掉含游標所在那行的以後全部。

x　　殺掉游標所在的那個字母。

3x　　殺掉含游標所在的那個字母的以下 3 個字母。

X　　殺掉游標所在的前一個字母。

3X　　殺掉含游標所在的那個字母的以前 3 個字母。

u　　Undo！回復到剛剛編輯前的狀態。

U　　在還沒離開本行之前，回復所有在這一行的編輯。

另外值得一提的是，在你用上述命令殺掉東西時，殺掉的東西實際上會保留在一個無形的 buffer 內，你可以將它再"貼"回來：用"p"（paste）這個命令。以前面那個例子來說：

```
I love UNIX and I hate vi.
Learning vi is a bad thing.
Learning vi is a good thing.
```

假設你現在用"d5w"將"vi is a bad thing"殺掉，然後，你將按"p"，你會發現"vi is a bad thing"這幾個被殺掉的字被"貼"回來了，貼的位置是在游標的後面。如果你要它貼在游標的前面，那就用大寫"P"。你把游標移動到最後一行，然後按"P"看看它的效果就知道了。這就是一般編輯器的"cut and paste"功能。

那你又要問了，"copy and paste"呢？被殺掉的字有放在那個無形的 buffer 內，可不可以 copy 一些東西進去那個無形的 buffer 內呢？也是可以的，"copy"在"vi"的術語叫"yank"（爲什麼要叫"yank"呢？好問題！"yank"是"拔出"的意思，像拔牙齒就用："yank out a tooth"，用它來表示"copy"東西到buffer 內，大概不太符合我們老中的語言習慣，爲什麼要叫"yank"呢？我也不知道。但是經過我這樣囉唆一段，你大概可以記住，"yank"在"vi"裡的意思了。）

現在假設你要將"love UNIX"拷貝到最後一行，你先將游標定到"love"的"l"上，然後按"y2w"（yank out 2 words），這時"love UNIX"就被拷貝到那個無形的 buffer 內了，然後你再將游標移動到最後一行，你要放"love UNIX"的地方，按"p"或"P"，"love UNIX"就出現了！

下面把幾個做這類編輯的命令列出來：

yy　　或"y1y"、"1yy"，yank out a line，將游標所在那行拷貝到 buffer 內。

y3y　　或"3yy"，yank out 3 lines，拷貝3行，含游標所在那行。

y4w　　yank out 4 words，拷貝 4 個字，含游標所在那個字。

p　　paste。將 buffer 的內容貼在游標的後面。

P	Paste。將 buffer 的內容貼在游標的前面。

再配合前面的"d"命令，你要的基本編輯功能就完備了。

經過一些練習，你一定有些受不了游標移動的方法了。先不論你到底熟悉 HJKL 四兄弟了沒，一次一個字母，或一次一行的移動方式，想必也會讓人抓狂。現在來教大家快速移動遊標，要到哪裡就定到哪裡，只要一個步驟，絕不囉唆！先把這些移動游標的命令列出來：

H	游標向左。
L	游標向右。
J	游標向下。
K	游標向上。
w	"w"是"word"的縮寫，跳到下一個字（的頭一個字母）。
3w	跳到下第三個字（的頭一個字母）。
b	"b"是"backward"的縮寫，跳到前一個字（的頭一個字母）。
3b	跳到前第三個字（的頭一個字母）。
e	"end of word"的意思，跳到游標所在這個字的最後一個字母。
Control-B	"backward"的縮寫，往上一頁。
Control-F	"foreward"的縮寫，往下一頁。
G	到檔案最後。
0	跳到本行第一格。
$	跳到本行最後。

這幾個，你大概就夠用了。當然還不只這些，但是勸你還是先不要學，以免消化不良，反而學不來。

6.1.4 ed 模式下的 vi

說了這麼多，你會發現上面講的幾乎都是在"命令"模式下動作，只要你記得要按<ESC>，學"vi"就很簡單。

之前說到"vi"還有個"ed 模式"，截至目前爲止，除了最開始講的":w"（存檔案）和":q"（跳出 vi）之外，還沒看到它的出現，意思很明顯：不會"ed 模式"也不要緊，會了就更好。現在來看看在"ed 模式"下，有什麼好用的。

只要你在"命令"模式下先按":"（冒號），就是進入"ed 模式"。爲什麼叫做"ed模式"呢？因爲在":"所下的命令，都是從"ed"這個編輯器來的。"ed"算是"vi"的前身，是一個"行編輯器"，由它衍生出來的還有"sed"（Stream Editor），也是一個很好用的程式，在後面章節會做介紹。

現在先把"ed模式"下的幾個重要命令列出來：

:w	存（"write"）檔案。可以在":w"後加上檔案名稱，表示要將它寫出到該檔案。
:r 檔名	"read"，將一個檔案讀進來，放在游標的位置。相當於一般編輯器的"include"功能。
:q	跳出（"quit"、離開）"vi"。可以用":q!"強制離開。
:1	將游標跳到第 1 行。
:12	將游標跳到第 12 行。
:$	將游標跳到最後一行。跟"G"命令一樣。
:/keyword	尋找"keyword"這個字串。可以省略":"，直接在命令模式下打："/keyword"。
:4,12s/ABC/DEF/	將第 4 行到第 12 行中的第一個"ABC"換成"DEF"。最後一個"/"可以省略。
:4,12s/ABC/DEF/g	將第 4 行到第 12 行中的所有"ABC"都換成"DEF"。"4,12"表示範圍，"s"表示"search and replace"，"ABC"是要尋找的關鍵字，"DEF"是要換成什麼字串。"g"表示"globally"，在同一行內如果該關鍵字出現不只一次，也都要換掉。如果沒有"g"，則只換在那一行出現的第一個。
:1,$s/ABC/DEF/	將第 1 行到最後一行中（就是整個檔案啦！）的第一個"ABC"，換成"DEF"。最後一個"/"可以省略。
:1,$s/ABC/DEF/g	將第 1 行到最後一行中的所有"ABC"，換成"DEF"。
:! ls -la	執行一般殼的指令"ls -la"，再回到"vi"。因爲"vi"會佔據整個殼，所以你就沒辦法再給指令（在這個殼內），你可以在 ":"之後加個"！"，然後給殼的指令，這樣就可以在不跳出"vi"的情況下暫時執行殼的指令。

這幾個就夠了。其中有些看起來很奇怪的表示法（如":4,12s/ABC/DEF/"）牽涉到 *UNIX* 的所謂"regular expression"，這個東西在很多指令都會用到，在下一節我們還會介紹。

好，差不多了。"vi"眞的不那麼難學，用習慣了你眞的會愛上它。

最後，再補充幾個命令（在命令模式）：

n	前面說用"/"來搜尋一個關鍵字，找到的時候，游標會停在那個關鍵字上，當你要再往下（檔案後面）搜尋下一個該關鍵字時，你只要按"n"就行了。
N	如果要往前再搜尋，按"N"。
ZZ	跟":wq"一樣意思。
.	不管上個編輯命令是什麼，重複上個命令。

另外，在"vi"內要輸入控制字元如"Control-D"、"Control-G"等等，要先按"Control-V"然後再按該字元。例如要輸入"^D"：當然，先進入輸入模式，再按住 Control 鍵，先按"V"再按"D"，就成了。

眞的講完了，學不學在你，講得不清楚的地方請你一定要告訴我...。

6.2 sed 和 Regular Expression

在繼續讀下去之前，先告訴各位讀者：以下這段將要教你熟悉使用所謂的"regular expression"，我會搭配"sed"這個程式來說明，並舉例說明它的用法，內容絕對易懂好消化，請各位耐心看。 我的目的是要讓大家都能好好學會簡單的"regular expression"，不再像無字天書一樣，盡讓一堆奇怪的符號給搞糊塗了。會了"regular expression"，以後你用 ed，sed，awk，grep，vi，等程式時，就會非常得心應手，離"高手"的日子也就不遠了。

以下的例子，我用的是這個檔案[3]：

```
There are three distinct versions of time: it is built in to
the  C  shell,  and  is  an  executable program available in
/usr/bin/time  and  /usr/5bin/time  when  using  the  Bourne
shell.   In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

姑且將它命名爲"unix.txt"。

6.2.1 Find and Replace

先說說"sed"這個程式的普通用法。簡單說，就是來代換字串、字元。例如：

```
% sed 's/time/TIME/' unix.txt
```

還記得在講 C Shell 時提到":s"，代換[4]嗎？上面這個例子的"s"同樣表示"search"的意思：對"unix.txt"這個檔案的每一行做動作。所以，上面例子是說，將檔案中的"time"換成"TIME"，結果是：

```
There are three distinct versions of TIME: it is built in to
the  C  shell,  and  is  an  executable program available in
/usr/bin/TIME  and  /usr/5bin/time  when  using  the  Bourne
shell.   In each case, TIMEs are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

你發現，"/usr/5bin/time"的"time"沒被代換到。那是因爲你沒有說要將一行中的"每一個"被找到的字串代換掉，所以"sed"只對每一行中的第一個"time"做動作。"/usr/5bin/time"的"time"

[3]這是 SunOS 裡面，"man time" 的一小段輸出。
[4]參閱第四章。

之前已有"/usr/bin/time"，所以只做"/usr/bin/TIME"。如果你是要對一行中，所有匹配到的字串做動作，那得在後面再加個"g"：

```
% sed 's/time/TIME/g' unix.txt
```

回憶一下在 C Shell 那一章，":s"和":gs"也是一樣道理。

這是"sed"最普通的用法，下面我們來看看更有彈性的匹配字元方式。

6.2.2 匹配行首：^

首先講到"行首"（Beginning of the Line）。先來看個例子：

```
% sed 's/^t/T/' unix.txt > unix.txt2
```

這個指令是說：對"unix.txt"這個檔案的每一行尋找"^t"，然後將它換成"T"。因爲在"regular expression"中的"^"是表示"行首"，所以這個指令就變成：將"unix.txt"這個檔案中，以"t"開頭的行，將"t"換成"T"。所以經過這道指令以後，"unix.txt2"的內容應該是：

```
There are three distinct versions of time: it is built in to
The C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

注意，第二行"the C Shell"的"the"被換成"The"了，但第一行"three distinct"的"three"中的"t"並沒有被代換，因爲它不是在行首。

做個測驗：下面這道指令的結果，你能先預測嗎？

```
% sed 's/^shell/SHELL/' unix.txt > unix.txt2
```

6.2.3 匹配行末：$

再來說到如何表示"行末"，一行的最後（End of the Line）：用"$" 符號（記得嗎？在"vi"中，也是用"$"來將游標跳到行尾的）。 先看例子：

```
% sed 's/in$/IN/' unix.txt > unix.txt2
```

這個例子是說：將"unix.txt"這個檔案中，以"in"結尾的行，將"in"改成"IN"。所以，"unix.txt2"的內容應該是（注意第二行行尾的 IN）：

```
There are three distinct versions of time: it is built in to
the C shell, and is an executable program available IN
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

注意第三行"using"的"in"並沒有被代換，因爲它不在一行的最後。

接著，來看看 Regular Expression 的其它符號。

6.2.4 匹配任一字元：.

"." 代表"任一字元"，就跟在 C Shell 中，"？"代表"任一字元"一樣。"..."則代表任意三個字元所組成的字串，如"abc"，"#␣7" [5]等等。看個例子：

```
% sed 's/e.e/HERE.HERE/g' unix.txt
ThHERE.HERE are three distinct versions of time: it is built in to
the C shell, and is an HERE.HEREcutable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

這個指令是尋找"e.e"，就是兩個"e"中間夾一個字元的。結果找到兩個地方，將它們換成"HERE.HERE"。特別注意的是，我們講匹配的方法說的"."、"^"（行首）、"$"（行尾），都只對"搜尋"有效，在"代換"時，這三個符號並沒有特別的意義，點就是點，錢號就是錢號。所以在這個例子中，找到"e.e"這樣的組合有："There"，和"executable"，代換成"HERE.HERE"，夾在"HERE"中的"."，就是指"."，不再是搜尋時的"任一字元"的意思。

想一想，試一試，下面三個指令的結果是什麼：

```
% sed 's/...ee/33$^333/g' unix.txt
% sed 's/^/^/' unix.txt
% sed 's/$/$/' unix.txt
% sed 's/^$//' unix.txt
```

注意最後一個指令，它是將檔案中的空白行除去，你知道爲什麼嗎？

6.2.5 多種選擇的匹配法：[...]

接著，再介紹如何有多個選擇，比如你要選"以阿拉伯數字開頭的字"、"e 的前面是 T 或 A 或 r 的字"等等。用的方法是跟在 C Shell 一樣：中括號"[...]"。如，[abc]表示是"a"，或"b"，或"c"（一個字母，不是"abc"哦）；[0-9]表示任一阿拉伯數字；[A-Za-z]則表示任一大寫或小寫字母。看個例子再談：

```
% sed 's/[Tt]he/THE/g' unix.txt
THEre are three distinct versions of time: it is built in to
THE C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using THE Bourne
shell. In each case, times are displayed on THE diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

[5]本書中的 ␣ 符號表示一個空格（space bar）。

這個例子是尋找"the"或"The"，將它換成"THE"，"g"表示如果在一行內有找到超出一個，也全部代換。再看：

```
% sed 's/^[tT]he/THE/g' unix.txt
THEre are three distinct versions of time: it is built in to
THE  C  shell,  and  is  an  executable program available in
/usr/bin/time  and  /usr/5bin/time  when  using  the  Bourne
shell.   In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

和前一個例子相比，只差在尋找的樣式多了一個"^"表示，在每行開頭的才算，所以不在行首的"the"或"The"就不被作用。另外在這個例子中，"g"有沒有其實是一樣，因爲"行首"在一行中一定是一個，不會有兩個以上。再看：

```
% sed 's/[0-9]/[0-9]/g' unix.txt
There are three distinct versions of time: it is built in to
the  C  shell,  and  is  an  executable program available in
/usr/bin/time  and  /usr/[0-9]bin/time  when  using  the  Bourne
shell.   In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

這個例子是尋找每一個阿拉伯數字，將它換成"[0-9]"這個字串，注意同樣的"[0-9]"出現在"搜尋"與"代換"的位置，有不一樣的解釋。

6.2.6 匹配零或多個字元：*

前面說的都是匹配一個字元的尋找方式，現在來看看如何表示多個字元。用"*"號來表示零或多個字元。例如，"A*"表示"零個，一個，兩個，三個... 大寫 A"。所以，"AA*"表示"一個或更多A"；"␣*"表示"零個或以上的空白"。

來看個例子：

```
% sed 's/e*c/EEE/g' unix.txt
There are three distinEEEt versions of time: it is built in to
the  C  shell,  and  is  an  exEEEutable program available in
/usr/bin/time  and  /usr/5bin/time  when  using  the  Bourne
shell.   In eaEEEh EEEase, times are displayed on the diagnostiEEE
output stream.

And, this is for spEEEial EEEases: [Test], -Test-, and {this}.
```

這個例子是說，找"零個或一個、更多e之後，接著是c的字串，將它用EEE取代"。檔案中，共找到了"distinct"（零個e之後，接著是c），"executable"（一個e之後，接著是c），"each"（零個e之後，接著是c），"case"（零個e之後，接著是c），"diagnostic"（零個e之後，接著是c），"special"（一個e之後，接著是c）。

再看：

```
% sed 's/ec*c/EEE/g' unix.txt
There are three distinct versions of time: it is built in to
the C shell, and is an exEEEutable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic
output stream.

And, this is for spEEEial cases: [Test], -Test-, and {this}.
```

這次只找到"executable"和"special"，因爲只有它們符合"e 之後有零個或一個、更多 c，然後再一個 c"的搜尋原則。注意，要將"ec*c"看成"e、c*、c"，"*"符號要綁著前一個字元（不多不少，就是一個），不可分開，就不會搞糊塗了。試試看下面這個例子：

```
% sed 's/␣␣*/␣/g'
```

是將檔案中，"一個以上"的空格，換成"一個"空格。

到目前爲止，你已經學過好幾個符號了，現在來考驗一下你所學的：

```
% sed 's/e.*e/#####/g' unix.txt
Th#####: it is built in to
th##### in
/usr/bin/tim#####
sh##### diagnostic
output stream.

And, this is for sp#####st-, and {this}.
```

這個結果想必嚇了你一跳吧！慢慢解釋給你聽。首先，我們來看看"e.*e"怎麼拆解；根據剛才說的，"*"永遠跟著前一個字元，不要分開，所以，就變成："e 、.* 、e"，"e"當然沒問題，就是字母"e"，沒什麼大不了。有問題的是".*"。依照前面說的，"."有特殊意義，表示"任一字元"，"*"表示"前一字元出現零、或以上"次數。所以，".*"就變成"任一字元出現零、或以上次數"的字串，"e.*e"就成了："兩個字母 e 之間，沒有字元、或有出現任何個數的字元"，然後將找到的字串換成"#####"。

回頭看看原檔案：

```
There are three distinct versions of time: it is built in to
the C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

了解了嗎？

有另一個疑問是，整個檔案有很多"兩個字母 e 之間，沒有字元、或有出現任何個數的字元"的字串，例如，在第一行就有"There are"、"are three"等等，爲什麼，卻只挑最長的那個？答案就是了，"最長的那個"。當系統找到能匹配的字串有很多時，它只選最長的那個，所以"There are"、"are three"都沒被選到，因爲它們都沒有"There are three distinct versions of time"來得長。

這是"regular expression"的另一個規則：挑最長的！

6.2.7 非常奇怪的表示法

接著看最後兩個很奇怪的匹配方法，乍看之下保證讓你退避三舍，例如"A\{1,7\}"、"\(.*\)"。如果你已經被嚇倒了，先別緊張，下面的說明我有把握讓你輕鬆學會它們。

先來看第一種：\{min,max\}（如："A\{1,7\}"）。

前面提到用"AA*"來表示"一個或以上的A"，在這裡則更有彈性，指定要找"1 到 7 個A"。同樣將"\{"和"\}"當成不可分的。中間的兩個數字表示一個範圍，例如"1,4"、"3,5"等等。先看例子：

```
% sed 's/e\{2,3\}/####/g' unix.txt
There are thr#### distinct versions of time: it is built in to
the  C  shell,  and  is  an  executable program available in
/usr/bin/time  and  /usr/5bin/time  when  using  the  Bourne
shell.   In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

這個例子說，找"e\{2,3\}"，就是"2 到 3 個e"的字串，將它換成"####"。結果找到"three"，有兩個"e"。再看：

```
% sed 's/[A-Za-z]\{5,9\}/####/g' unix.txt
#### are #### #### #### of time: it is #### in to
the  C  ####,  and  is  an  ####e #### #### in
/usr/bin/time  and  /usr/5bin/time  when  ####  the  ####
####.   In each case, #### are #### on the ####c
#### ####.

And, this is for #### ####: [Test], -Test-, and {this}.
```

這個例子利用到前面說的"[A-Za-z]"，表示任意一個字母（大小寫都算）。所以"[A-Za-z]\{5,9\}"就是表示"字母個數在 5 到 9 之間的字串"。注意，"diagnostic"有 10 個字母，所以只被代換掉前 9 個，第十個字母"c"就沒有被作用。再來：

```
% sed 's/[A-Za-z]\{5\}/####/g' unix.txt
#### are #### ####nct ####ons of time: it is #### in to
the  C  ####,  and  is  an  ######## ####am ####able in
/usr/bin/time  and  /usr/5bin/time  when  ####  the  ####e
####.   In each case, #### are ####ayed on the ########
####t ####m.

And, this is for ####al ####: [Test], -Test-, and {this}.
```

這個例子中，只給一個"5"，沒有範圍，表示"正好 5 個"。所以："There"，"three"整個換成"####"，"distinct"的前 5 個字母也算，剩下的"nct"沒有動。其它類推。再來：

```
% sed 's/[A-Za-z]\{5,\}/####/g' unix.txt
```

```
#### are #### #### #### of time: it is #### in to
the C ####, and is an #### #### #### in
/usr/bin/time and /usr/5bin/time when #### the ####
####. In each case, #### are #### on the ####
#### ####.

And, this is for #### ####: [Test], -Test-, and {this}.
```

這個例子在 5 的後面有“,”，但沒有第二個數字，表示“5 個以上”。所以，所有含“5 個字母或以上”的字串，都會被作用。注意，像“distinct” 這樣，含 10 個字母的，也符合“5 個字母或以上”的匹配，所以它用最長的匹配：10 個。將 10 個字母都換成“####”。

利用“\{min,max\}”，再配合之前說的各種匹配法則，你可以隨意試著組合，看看效果如何，多玩幾次就會了。

最後還有一個，嚴格來說，不是搜尋時的匹配方法，只是用來儲存被匹配到的字串：\(匹配法\)。同樣把“\(”跟“\)”看成不可分割的。先看一個例子：

```
% sed 's/\([tT]he\)/\1+++++/g' unix.txt
The+++++re are three distinct versions of time: it is built in to
the+++++ C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the+++++ Bourne
shell. In each case, times are displayed on the+++++ diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

這個例子中，在“\(”跟“\)”之間的是前面說的匹配法之一：“[tT]he”，就是找到“the”或“The”，然後將它暫時存在代號 1 號的黑盒子內，在它的後面加上“+++++”，再填回去。這裡的“\1”用來表示被找到的東西（根據在“\(”跟“\)”之間的匹配法則所找到的東西）。所以，第一行找到“The”，“\1”就是“The”；第二行找到“the”，那該行的“\1”就是“the”。再看：

```
% sed 's/\(.*\) \(.*\)/\2 \1/' unix.txt
to There are three distinct versions of time: it is built in
in the C shell, and is an executable program available
Bourne /usr/bin/time and /usr/5bin/time when using the
diagnostic shell. In each case, times are displayed on the
stream. output

this. And, this is for special cases: [Test], -Test-, and
```

這個例子的匹配法是：\(.*\) \(.*\)，有兩對“\(”和“\)”，所以除了“\1”外，還會有“\2”。而其中的匹配法是“.*”，表示是任一字元出現零次或多次；把“\(”和“\)” 先拿掉的話是：“.* .*”，表示以空格隔開的兩個字串。

所以，：\(.*\) \(.*\) 就是把空格兩邊的字串分別存在 1 號，2 號盒子內。代換的方法是：“\2 \1”，就是將兩個互調，原來放在 2 號盒子內的東西，擺到 1 號前面，中間一樣以空格相隔（“\2”和“\1”間有個空白鍵）。所搜尋的結果是：

There are three distinct versions of time: it is built in to
the C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic

output stream.

And, this is for special cases: [Test], -Test-, and {this}.
還是一樣，先找到最長的，放在 1 號，剩下的，就是每行最後一個字，放在 2 號。再對調，就是結果了。

這個用法比較奇怪，如果你還沒看懂，建議你再仔細看一遍。如果看了三遍還沒懂，請你放棄不要學了，不是嫌你笨，是我沒講清楚，再看也沒用，更何況這個用法不會也不要緊。

最後要補充說明的是，在前面說的匹配法中，用到的特殊字元如“.”、“*”等，如果要回復它們本來的意思，例如，你要尋找的就是“.”這個字元，而不是“任一字元”，那就要用“\”來“避開”。來看個例子：

```
% sed 's/\./$/g' unix.txt
There are three distinct versions of time: it is built in to
the C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell$  In each case, times are displayed on the diagnostic
output stream$

And, this is for special cases: [Test], -Test-, and {this}$
```

這個指令尋找的是句點“.”，然後將它換成“$”。注意“\.”，而不只有“.”，意義不同！

再看個例子：

```
% sed 's/\//%/g' unix.txt
There are three distinct versions of time: it is built in to
the C shell, and is an executable program available in
%usr%bin%time and %usr%5bin%time when using the Bourne
shell.  In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

這是將檔案中的“/”換成“%”。注意“\/”，是用“\”來避開代表“搜尋“意義的“/”。簡單的說，碰到可能有特殊意義的字母時，就在它前面加上“\”就對了，即使它不具有特殊意義也無妨，例如：

```
% sed 's/\T/AAAAAAAAAAAAAAAAAAA/' unix.txt
AAAAAAAAAAAAAAAAAAAhere are three distinct versions of time: it is built
in to
the C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell.  In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [AAAAAAAAAAAAAAAAAAAest], -Test-, and {this}.
```

在“T”前面加上倒斜線跟不加是一樣的。

另外，照前面說的，“[A-Z]”是表示任一大寫字母，如果要尋找的是“任一大寫字母或-（減號)”，要用“[-A-Z]”，不可以用“[A-Z-]”，也就是說，“-”要馬上跟在“[”之後。還有：

```
% sed 's/[[ABC]/$$/g' unix.txt
```

```
There are three distinct versions of time: it is built in to
the $$ shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the $$ourne
shell.  In each case, times are displayed on the diagnostic
output stream.

$$nd, this is for special cases: $$Test], -Test-, and {this}.
```

這是匹配"["，或"A"或"B"，或"C"。寫成"[ABC[]"的話，"["就不會被找到。搜尋字元"["或"]"要緊跟在表示範圍的"["（左括號）後面。除非是要匹配行首"^"，那麼上兩例的"-"和"["就要在"^"之後，除此外，要緊跟"表示範圍的"["（左括號）後面。

"regular expression"就這樣了，順便也把"sed"做了一些介紹，下面再說一點"sed" 的應用。

6.3 關於 sed 的其它用法

前面說到的'sed"都用來做簡單的"search and replace"編輯動作，現在來看看幾個常見的"sed"應用。

在前面說到的代換動作中，都是對檔案中的"每一行"作用，如果你是要對特定的幾行做動作，就要告訴"sed"是哪幾行：

```
% sed '1,6s/the/THE/g' unix.txt
```

這個例子是將"unix.txt"這個檔案中第一行到第 6 行中的"the"換成"THE"。注意這個例子和前面說的只在"s"之前加上"1,6"而已。

如果是要對含有某特定字的行作用，則要用：

```
% sed '/shell/s/the/THE/g' unix.txt
```

這是說，在"unix.txt" 檔案中，含有"shell" 這個字眼的那些行中，將"the" 換成"THE"。"/s/the/THE/g"都和前面講的一樣，只是現在要對含有"shell"這個字眼的那些行作用，所以在"s"之前先搜尋一次："/shell"，就像在"vi"中搜尋關鍵字一樣用"/"！

"sed"可以用來殺掉某些行：

```
% sed '3,5 d' unix.txt
There are three distinct versions of time: it is built in to
the  C  shell,  and  is  an  executable program available in

And, this is for special cases: [Test], -Test-, and {this}.
```

這是將檔案的第 3 到第 5 行殺掉的意思。"d"表示"delete"，大小寫都可以，數字和字母間也可以不加空格。如果只要殺掉一行，兩個數字一樣，或只給該行的行號就可以：

```
% sed '3,3 d' unix.txt
      或
% sed '3 d' unix.txt
```

也可以將含有關鍵字的行殺掉：

```
% sed '/shell/d' unix.txt
There are three distinct versions of time: it is built in to
/usr/bin/time and /usr/5bin/time when using the Bourne
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

這是將含有"shell"字眼的行去掉的意思。

除了代換和"delete"的功能外，"sed"還會從一個檔案中挑出特定的幾行：

```
% sed -n '3,5 p' unix.txt > 3-5.txt
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic
output stream.
```

這是將檔案的第 3 行到第 5 行抽出來，放到"3-5.txt"這個檔案中。"p"表示"print，大小寫都可以，和數字之間沒有空格也不要緊。但，千萬不要忘了"-n"這個 option！到目前爲止，這是唯一要加選項的"sed"指令。

```
% sed -n '/shell/p' unix.txt
the C shell, and is an executable program available in
shell. In each case, times are displayed on the diagnostic
```

這是把含有"shell"字樣的行抽出來。對照"d"，也不要忘了"-n" ！！

好了，"sed"在這裡告一段落，學會這些就夠了，其它更高竿的用法就請各位再找專門的書看吧。

6.4 簡介 grep 指令

接著來看另外一個利用"regular expression"的指令："grep"。它最常用的是：

```
% grep shell unix.txt
the C shell, and is an executable program available in
shell. In each case, times are displayed on the diagnostic
```

就是把含有"shell"的那些行印出來。這個效果跟前一例用"sed"相等。但是，當你想在一堆檔案中找出到底哪些檔案有含某個關鍵字時，就只能用"grep" 了：

```
% grep shell *
tools.tex:shell. In each case, times are displayed on the+++++ diagnostic
csh.manpage: Symbolic links can fool the shell. Setting the hardpaths
csh.manpage: To detect looping, the shell restricts the number of alias
intro.tex: full name ,shell。
```

這個例子可以看出"grep"把所有含"shell"字樣的那些行都挑出來，並且告訴你是在哪個檔案中。

在使用"grep"時，將要找的關鍵字用單引號包起來是個值得養成的好習慣，因爲這樣可以避免一些不必要的麻煩：

```
% grep * *
```

你的原意可能是：在目前目錄下的所有檔案（第二個星號的意義）中，找出含有星號"*"（第一個星號的意義）的那些行。但是在 C Shell 中，"*"表示"所有檔案"，所以在"grep"動作之前，聰明的殼早就將兩個"*"換成所有檔案的名字，"grep"當然不會正確作用。正確的作法是將"*"包在單引號"'"內：

```
% grep '*' *
```

注意喔，不要用雙引號，在 C Shell 中，單引號和雙引號的差別就在於，前者不會將特別字元做解釋，如"*"、"$"（變數名）等等；而後者卻會。

現在來考考各位"regular expression"：

```
% grep '[tT]he' unix.txt
```

想到答案了嗎？這是找含有"the" 或"The"的那些行的意思。再來：

```
% grep '\.' unix.txt
% grep '.' unix.txt
% grep . unix.txt
```

注意這三個指令的結果，想想看，爲什麼第三個指令是把檔案的空白行去掉？[6]

6.4.1 grep 的選項

接下來，我們來看看"grep"的選項有哪些好用的：

-i

```
% grep -i 'Shell' unix.txt
the  C  shell,  and  is  an  executable program available in
shell.   In each case, times are displayed on the diagnostic
```

"-i"選項是"ignore"的意思，就是"忽略大小寫"。本來找"Shell"是要大小寫完全一樣，加上"-i"就變"不管大小寫"了。

-v

```
% grep -v 'shell' unix.txt
There are three distinct versions of time: it is built in to
/usr/bin/time  and  /usr/5bin/time  when  using  the  Bourne
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

[6] 所有前面講過的"regular expression"的用法都可以搭配"grep"來使用，自己試試看，順便再溫習一下"regular expression"。

"-v"是"invert"的意思，就是"相反"，把"不"含有關鍵字的行抓出來。

-n

```
% grep -n 'shell' unix.txt
2:the  C  shell,  and  is  an  executable program available in
4:shell.   In each case, times are displayed on the diagnostic
```

"-n"是"line number"的意思，就是在抓到的每一行前面，標上該行的行號。下面的例子是將一個檔案的內容加上行號，與"cat -n"和"nl"指令有類似效果：

```
% grep -n ^  file
```

注意在行號後面有"："號，跟"cat -n"不一樣，你可以想辦法濾掉它。

-l

```
% grep -l 'shell' *
cmd.tex
csh.manpage
intro.tex
login.sample
```

"-l"是"list"的意思，就是"只把含有關鍵字的檔名列出來"，內容不列。

-e

```
% grep -e '-Test' -n unix.txt
7:And, this is for special cases: [Test], -Test-, and {this}.
```

這個例子比較特殊的是，所要尋找的關鍵字含有"-"，所以，當你用："grep '-Test'"時，會得到"illegal option -- T"的訊息，因爲"grep"把"-"當成是選項的前導符號。要讓"grep"了解這個"-""不是"選項，就用"-e"這個選項，這樣一來，"grep"就會把"-Test"看成一體，不會將"-T"看成選項。另外，若搭配其它選項時，"-e '-Test'"要看成一體，然後才加其它選項，所以"-n"不能插在"-e '-Test'"中間，以免意思不一樣。

"grep"就大概就說到這裡了，你可以用"man grep"再多學一些。

6.5 簡介 cut 的用法

接著說"cut"指令。

學"cut"指令，你只要記住三個選項就行了："-c"（Character）、"-d"（delimiter）、"-f"（field）。現在先看個例子：

```
% cut -c1-7 unix.txt
There a
the  C
/usr/bi
```

```
shell.
output

And, th
```

這是說，將"unix.txt"這個檔案中"每一行"的第"1-7"個"字母(character)"切出來。"-c"表示"切字母"，後面跟著一個範圍（1-7）的數字。

```
% cut -c-7 unix.txt
```

這個指令跟上個指令完全一樣，當起始字元位置沒有給的時候，就是"1"的意思。

```
% cut -c7- unix.txt
are three distinct versions of time: it is built in to
  shell,  and  is  an  executable program available in
in/time  and  /usr/5bin/time  when  using  the  Bourne
   In each case, times are displayed on the diagnostic
 stream.

his is for special cases: [Test], -Test-, and {this}.
```

這個指令沒有給終點字元位置，所以，就是"一直到該行最後一個字元"。

```
% cut -c7 unix.txt
```

只給一個"7"，就是指定要第7個字元的意思。

或是跳著抓出某些特定位置的字元：

```
% cut -c1,3-7 unix.txt
Tere a
te  C
/sr/bi
sell.
otput

Ad, th
```

這是切出幾個字元的用法。但是有時候，檔案的樣子並不那麼規矩，都對齊得很好，可以讓你數得出數目，用"cut -c"的方式抓出你要的東西，例如：

```
john:ACrBqjTb8JmxQ:149:200:John Cooley:/home/users/john:/bin/csh
cpy:,xcrggtKWJTUA:172:200:C. P. Y:/home/users/cpy:/bin/tcsh
math:po.isgMVOPEw:110:200:Guess Who?:/home/users/math:/bin/sh
smith:ouzoi1EiDuJuk:129:200:Bob Smith:/home/users/smith:/bin/csh
student1:ou..41EiDuJuk:159:300:Talking Head:/home/users/student1:/bin/csh
```

這是一個典型的"/etc/passwd"檔案。假設你現在要抓出所有使用者的"full name"，你會發現用"cut -c"的方式根本行不通，因爲它們所在每行的位置都不一樣，但是它們都在每行的第 5 個欄位（field），以":"間隔。這種情況下，你可以用：

```
% cut -f5 -d':' /etc/passwd
```

```
John Cooley
C. P. Y
Guess Who?
Bob Smith
Talking Head
```

"-f"和"-d"是要搭配用的。這個例子是說：切出"第 5 個欄位"（-f5），欄位定義是用":"來分隔的（-d':' ）。

當然，你也可以一次抓出多個欄位：

```
% cut -f1,5 -d':' /etc/passwd
john:John Cooley
cpy:C. P. Y
math:Guess Who?
smith:Bob Smith
student1:Talking Head
```

6.6 順便談談 paste

接著也順便談一下"paste"指令。

"paste"指令是用來將兩個檔案"肩併肩"貼在一起的，先看個例子：假設有兩個檔案，各叫"p1"、"p2"：

```
% cat p1
Line 1
Line 2
Line 3
Line 4
% cat p2
L1
L2
L3
L4
L5
L6
```

現在將它們"paste"起來：

```
% paste p1 p2
Line 1  L1
Line 2  L2
Line 3  L3
Line 4  L4
        L5
        L6
```

貼起來的檔案裡，原先兩個檔案的每一行之間，都用<TAB>鍵分隔，所以第一行實際上是 "Line 1<TAB>L1"；如果你想用別的分隔符號，可以用"-d"指令，跟"cut" 一樣：

```
% paste -d':' p1 p2
Line 1:L1
Line 2:L2
Line 3:L3
Line 4:L4
:L5
:L6
:
```

說"paste"是對"兩個"檔案作用是不太正確的，因爲它事實上可以只對一個檔案做"paste"的動作，只是這時候就要加個"-s"的選項了：

```
% paste -s p1
Line 1  Line 2  Line 3  Line 4
```

"-s"是"serial"的意思，就是說將一個檔案的每一行通通接起來，變成一行。相當於將檔案中的"跳行字元"換成<TAB>。當然，你也可以換成其它的分隔字元，用前面說的"-d"選項：

```
% paste -s -d':' p1
Line 1:Line 2:Line 3:Line 4:
```

6.7 還有 tr

再來，我們來學"tr"指令。

"tr"是"translate"的縮寫，顧名思義，是將某個字元轉（換）成另一個字元。先看個例子：

```
% tr T @ < unix.txt
@here are three distinct versions of time: it is built in to
the  C  shell,  and  is  an  executable program available in
/usr/bin/time  and  /usr/5bin/time  when  using  the  Bourne
shell.   In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [@est], -@est-, and {this}.
```

這個例子是說：把"unix.txt"這個檔案中的"T"都轉成"@"。注意"tr"指令的既定輸入（default input）是標準輸入（standard input），如果你要轉的是一個檔案，要用"<"來"重導"。

"tr"指令不只可以轉一般的字元、字母，它可以處理其它特殊字元，例如：跳行字元（Newline）、退位字元（Tab）等等，在轉這些特殊字元時，可以用"\ascii_number"的方式來表示：

```
% tr '-' '\12' < unix.txt
There are three distinct versions of time: it is built in to
the  C  shell,  and  is  an  executable program available in
/usr/bin/time  and  /usr/5bin/time  when  using  the  Bourne
shell.   In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test],
```

```
Test
, and {this}.
```

注意，檔案中的"-"已經被換成跳行了，"\12"是"Newline"的 ASCII 碼。你可以用"man ascii"來查 ASCII 表，找到你要的特殊字元的 ASCII 碼。下面是幾個常用的：

Beep，嗶聲	007
Backspace	010
TAB	011
Newline	012
Escape	033

"tr"也可以轉一個範圍的字元，例如：

```
% tr '[a-z]' '[A-Z]' < unix.txt
THERE ARE THREE DISTINCT VERSIONS OF TIME: IT IS BUILT IN TO
THE  C  SHELL,  AND  IS  AN  EXECUTABLE PROGRAM AVAILABLE IN
/USR/BIN/TIME  AND  /USR/5BIN/TIME  WHEN  USING  THE  BOURNE
SHELL.   IN EACH CASE, TIMES ARE DISPLAYED ON THE DIAGNOSTIC
OUTPUT STREAM.

AND, THIS IS FOR SPECIAL CASES: [TEST], -TEST-, AND {THIS}.
```

這個例子把小寫字都換成大寫字了。要注意兩個範圍的一致性，這個例子的"a-z"有 26 個字元，而"A-Z"也有 26 個，所以可以一對一的換。你可以試試看把"[A-Z]"變成"[0-9]"，就知道當"tr"找不到該對應的字元時，後果是什麼。

"tr"有兩個重要的選項非學不可，第一個是"-s"，是"squeeze"的意思，顧名思義，就是會"擠壓"的意思，"擠壓"什麼呢？就是把多個重複的字元在"translate"過程中變成一個：

```
% head -1 unix.txt | tr -s 'e' 'E'
ThErE arE thrE distinct vErsions of timE: it is built in to
```

這個例子的前段是先取"unix.txt"的第一行，將它送給"tr"做轉換，要將小寫"e"變成大寫"E"，在轉的過程中，要將多個"E"變成只有一個，所以原來的"three"就變成了"thrE"。

```
% tr -s ' ' ' ' < unix.txt
There are three distinct versions of time: it is built in to
the C shell, and is an executable program available in
/usr/bin/time and /usr/5bin/time when using the Bourne
shell. In each case, times are displayed on the diagnostic
output stream.

And, this is for special cases: [Test], -Test-, and {this}.
```

這個例子則是把多個空白，變成只有一個。

另一個要學的選項是"-d"，"delete"的意思，用來將某個字元去掉：

```
% tr -d ' ' < unix.txt
Therearethreedistinctversionsoftime:itisbuiltinto
theCshell,andisanexecutableprogramavailablein
/usr/bin/timeand/usr/5bin/timewhenusingtheBourne
shell.Ineachcase,timesaredisplayedonthediagnostic
outputstream.

And,thisisforspecialcases:[Test],-Test-,and{this}.
```

可以看到，空白都被去掉了，這跟下面指令是一樣的：

```
% sed 's/ //g' unix.txt
```

你現在應該會發現"tr"指令跟"sed"有很大程度上的功能重複，對的，你可以仔細比較看看，它們有哪些功能是對方無法取代的[7] 。

He said Mama, Mama
The President's a fool.
Why do I have to keep reading
These technical manuals ?

from Amused To Death
by Roger Waters

[7] "tr"只對一個字元有效，要轉一個字串的話...。

Chapter 7

學寫 C Shell Script

UNIX 的好用不只在一個一個獨立的程式和指令，更重要的是"殼"所扮演的角色。在第四章已經大致介紹了一些 C Shell 相當好用的技巧，在這一章，我們要進一步來看看，"殼"如何將這些技巧整合起來，結合一個個獨立的程式和指令做更有"學問"的事，在 DOS ，叫"批次檔"，在 *UNIX* 中，我們叫它"Shell Script"。

來學學寫 C Shell Script 吧[1] 。

7.1 你的第一個 Shell Script

還是先看個例子：

```
 1  #!/bin/csh
 2  set a = 1
 3  set b = 2
 4  echo a = $a
 5  echo b = $b
 6  if ( $a == $b ) then
 7      echo a == b
 8  else
 9      echo a \!= b
10  endif
```

這是一個很簡單的 shell script，第一行的"#!/bin/csh"在前面已經說明過，它告訴系統，這個檔案的內容是"C Shell"語言寫的，要用"/bin/csh"來執行以下每個指令。第二行和第三行分別設定兩個變數，a 和 b，其值各爲"1"、"2"。第四行和第五行則是把它們的值印出來，"echo"指令跟 C 語言的"printf"很像，都是把後面的東西印出來的意思，注意變數前的"$"號，是用來表示變數本身所存放的東西。第 6 到第 10 行是一個條件判斷式，"如果 a 的值等於 b 的值，則印出"a == b"這個訊息，否則，就印出"a != b"。注意第 9 行的"！"前要加上倒斜線"\"，是因爲"！"在 C Shell 中用來代表"history"，所以用倒斜線來回復它原來的意思。你可以把這個

[1] 在進行說明之前，先假設你曾寫過程式，有最基本的程式概念，C 語言、PASCAL 或 BASIC 等等都可以。雖然有人說 C Shell 不是一個好的程式語言，也缺乏有效的檔案處理功能，但它近似 C 語言的優點卻能讓初學 unix 的人快速上手，將 unix 玩得更得心應手。

小程式寫成一個檔案，假設命名爲"equal"，然後，記得將它的屬性改爲"可執行"，接著，執行它："./equal"。

特別一提，因爲第一行"#!/bin/csh"的緣故，系統知道是"C Shell"語言寫的，所以系統就會再挖一個"C 殼"給你去執行以下的東西，因爲是"C 殼"，所以，你的"～/.cshrc"又會被"看"（執行）一次。換言之，當這個程式要執行時，它事實上是在目前的"殼"之中，再產生一個"子殼"，執行完"～/.cshrc"，然後再來跑第二行以後的東西，等到都執行完了，再收起這個臨時產生的"子殼"，回到原來的"殼"，當它回到原來的"殼"時，就像沒發生過"子殼"一樣，因爲那個"子殼"已經不見了。除了在執行過程中留下的東西外（是你要它印的東西）：

```
a = 1
b = 2
a != b
```

這個程式夠簡單了吧，但是學到的東西可是不少："#!/bin/csh"，所代表的意義以及它所產生的後續動作（子殼）；"set"，設定變數（哪種程式語言不用變數的？）；"echo"，在適當時機印出你想看的訊息；"$"，取得變數的值（數字或字串）；"if ...then ...else ...endif"，注意"if"和"endif"的成對關係，還有，"then"要跟"if"同一行，避免錯誤發生。

有時候，程式執行的過程不如所願，這時就要"抓蟲"（debug），你可以適度的用"echo"。在程式的各個角落放一堆"echo"，這樣就可以監督每行執行的結果，以做修正。但是，還有更好的方法，就是在第一行"#!/bin/csh"的後面加上"-v"的選項，告訴它將每一步驟的執行過程都印出來：

```
#!/bin/csh -v
set a = 1
set b = 2
echo a = $a
echo b = $b
if ( $a == $b ) then
        echo a == b
else
        echo a \!= b
endif
```

加了"-v"之後，執行的結果：

```
set a = 1
set b = 2
echo a = $a
a = 1
echo b = $b
b = 2
if ( $a == $b ) then

echo a != b
a != b
endif
```

注意，輸出的東西多了程式要執行的每一行，以及所產生的結果。這些多出來的東西是"-v"選項的傑作。另外，前面說過，在進入"子殼"後，會先執行"～/.cshrc"，再執行其它內容，有時候你會嫌這樣太慢了，反正程式內又不必用到在"～/.cshrc"中的一些設定，執行它反而拖慢執行的速度，這時候你可以加上"-f"的選項，告訴它不必看"～/.cshrc"，直接執行內容。你現在可以試試看這加不加"-f"選項的效果各是如何。

記得那個有趣的指令叫"yes"的嗎？你可以寫一個跟它一樣功能的 script：

```
 1  #!/bin/csh -f
 2  if ( $#argv == 0 ) then # repeat printing 'y'
 3      while ( 1 )
 4            echo y
 5      end
 6  else   # repeat printing all arguments
 7      while ( 1 )
 8            echo $argv[*]
 9      end
10  endif
```

先介紹兩個變數："$#argv"以及"$argv[*]"。前者代表"number of arguments"，也就是"參數的個數"；後者代表"所有參數"。比如："ls .cshrc .login"這個指令的"參數的個數"是"2"，第一個參數是".cshrc"第二個參數是".login"；所有參數是".cshrc .login"。

現在來一行一行解釋這個有趣的 script 。第一行就別提了，如果你還不知道它的意思，請回頭看上一節。接著的第二行到第十行是一個條件判斷，注意第 2，6，10 的結構是"if ...then ...else ...endif"；在"if"這個判斷裡，我們先看看"參數的個數是不是零"，如果是，也就是說，在指令後面沒有給任何參數的話，就做第 3，4，5 行。現在看第 3，4，5 行是幹什麼的，"while"指令也是一個條件判斷，在"( )"內是一個條件，如果是"true"，就一直重複做以下的事，一直到"( )"內的條件不再滿足爲止，否則就跳過到"end"之後。注意"while"和"end"是成對的，就跟"if"要配一個"endif" 一樣。你看在"( )"內是"1"，就是"永遠爲眞"，所以就會一直做"echo y"這件事。整個看 2-5 行的意思就是："如果沒有給參數，就一直不停地印出 'y' "。

同理，第 6，7，8，9 行的意思是："否則（有參數，不管幾個），就一直重複把參數印出來"。

另外注意程式中的"#"字號，它代表註解，不做任何事。當然啦，第一行的"#!"是例外。

我們姑且把這個程式存成"my-yes"這個檔，記得把屬性改爲"executable" （例如：chmod +x my-yes）。執行結果：

```
% ./my-yes
y
y
y
^C
% ./my-yes I Love UNIX
I Love UNIX
I Love UNIX
I Love UNIX
^C
```

記得用"Control-C"將它停掉，要不然，憑幾個"while (1)"，就可以把機器弄瘋掉。這是一個自己寫的"yes"指令，希望你了解的是"參數"的應用以及"while"的用法。

這裡再列出一個表格，告訴你，有哪些"參數變數"可以使用：

$#argv	number of arguments
$argv[*]	all arguments
$argv	同上，all arguments
$*	同上，all arguments
$argv[1]	第一個參數， argument 1
$argv[2]	第二個參數， argument 2
$argv[n]	第 n 個參數， argument n
$argv[1-3]	第一到第三個參數，argument 1, argument 2, argument 3
$1 $2 $3	同上，表示第一到第三個參數
$argv[1-n]	第一到第 n 個參數，argument 1, argument 2, ... argument n
$1 $2 $3 .... $n	同上，表示第一到第 n 個參數
$0	第零個參數，就是程式（指令）本身；注意，不能用"$argv[0]"！
$argv[$#argv]	最後一個參數

再介紹一個例子：

```
1 #!/bin/csh -f
2 set number = ( 1 2 3 4 5 6 7 8 9 10 )
3 set cnt = 0
4 foreach n ( $number )
5     set cnt = `expr $cnt + $n`
6 end
7 echo "1+2+3+4+5+6+7+8+9+10=$cnt"
```

這個程式先定義一個叫"number"的變數，它是一個集合，裡頭的成員包括數字 1 到 10。然後，再將變數"cnt"的初始值先設爲"0"，接著在 4，5，6 行是一個以"foreach ...end"成對包起來的一段程式，"foreach"顧名思義，是針對每個集合成員做的動作。"foreach"的後面要接兩個東西，第一個是變數名字（"n"，這個名字只在"foreach ...end"之內有效），第二個是所有要動作的成員（"$number"）。做的動作是："set cnt = `expr $cnt + $n`"，這個式子就是一般在其它程式語言中的"cnt = cnt + n"。在 C Shell 中你可以借助"expr" 指令來做運算，注意要將"expr $cnt + $n"包在兩個" `"之間（凡是"指令" 的結果要放到變數中時，都要把指令包在兩個 ` 之間） 。最後一行再印出總和，注意雙引號中的"$cnt"，是變數的值，如果將最後一行改用單引號，那就不一樣了，你可以試試看。

所以整個程式做的事其實很簡單，就是將 1 到 10 十個數字加起來，印出總和，就如此而已。執行結果：

```
1+2+3+4+5+6+7+8+9+10=55
```

這個程式其實可以精簡一點：

```
#!/bin/csh -f
set cnt = 0
foreach n ( 1 2 3 4 5 6 7 8 9 10 )
       set cnt = `expr $cnt + $n`
end
echo "1+2+3+4+5+6+7+8+9+10=$cnt"
```

注意，我們將"set number = (....)"這一行拿掉了，直接把運算的成員填入"foreach"的括號內。

再看另一種寫法，一樣是要把 1 到 10 十個數字加起來：

```
#!/bin/csh -f
@ n = 1 # set n = 1
@ cnt = 0           # 同： set cnt = 0
while ( $n < 11 )
@ cnt += $n # 同： @ cnt = $cnt + $n 和：  set cnt = `expr $cnt + $n`
@ n += 1    # 同： @ n = $n + 1
end
echo "1+2+3+4+5+6+7+8+9+10=$cnt"
```

這個寫法利用的是"while"來做條件判斷，當變數 n 還沒變成 11 時，要一直做兩件事，一個是將變數 n 的值加入變數 cnt ，另一個動作是把變數 n 的值遞增一。

特別注意這裡的變數設定方法不再用"set"的方法；加法運算的方式不再利用"expr"指令，因爲在 C Shell 裡頭已經提供一個"@"運算，另外注意有"#"的那些行[2] ，都是用來做註解的，不會有作用。你可以試試用不同的運算法（用"expr"、不用"expr"；用"set"、不用"set"），比較看看執行速度的差異。

7.2 用 Shell Script 改檔名

再看個"foreach"的例子：

```
#!/bin/csh -f
foreach file ( *.ab)
        set base=`basename $file .ab`
        mv $file $base.cd
end
```

UNIX 裡頭並沒有像 DOS 中的"rename"指令，可以把一堆檔案一次改名，例如"ren *.ab *.cd"，如果你以爲用"mv *.ab *.cd"就可以，那肯定要沮喪萬分，因爲你忘了"*"的特殊意義。

你會說，堂堂 *UNIX* 竟然沒有 DOS 好用！好，我承認在這一丁點小事上， *UNIX* 的確輸給 DOS，但只要寫一點程式、一小段 script 就可以了。上面這個程式就是利用"foreach"把一堆名字是".ab" 結尾的檔案改成以".cd"結尾，達到"ren *.ab *.cd"的效果。注意"basename"這個指令[3] ，它用來取得一個檔案名字的前半段名字。

你又會問啦，小小一件事還得先寫個程式，存起來，改屬性爲"可執行"，然後再執行，太麻煩了。其實，像這種小程式，你不必寫成檔案再執行，直接告訴 C Shell 就可以了：

```
% foreach file ( *.ab)
? set base=`basename $file .ab`
? mv $file $base.cd
? end
%
```

注意，在你輸入"foreach"那行之後，C Shell 就知道你要寫東西給它做事情了，所以接下來的那行就自動會有"?"出現，等你寫東西進去，一直到你寫完"end"，C Shell 自然知道已經輸入完畢，

[2] 第一行除外。

[3] 請用"man basename"來看它的詳細用法。另外有個指令叫 dirname，可以順便學學。

就開始做事情了。透過這種方式，可以不必寫成檔案，再執行，但是缺點就是，無法重複使用這個 script，下回要做同一件事時，只好再輸入一次。

看最後一個例子：

```
 1 #!/bin/csh -f
 2 @ cnt = 1
 3 echo I will count from 1 to 100
 4 echo and pause at every ten numbers for confirmation to continue
 5 echo -n Press any key to start :
 6 set key = $<
 7 while ( $cnt < 101 )
 8         echo This is $cnt
 9         if ( `expr $cnt % 10` == 0 ) then
10            echo -n "Continue or not (Y/N):"
11            set key = $<
12            if ( $key == 'n' ) exit
13         endif
14         @ cnt += 1
15 end
16 echo DONE.
```

這個程式主要教大家一個由鍵盤讀取參數的方法："$<"。第 5 行的"echo -n"是告訴 C Shell 在印出後面的字以後，先不跳行，游標保持在同一行；接著的第 6 行是設定一個叫做"key"的變數，其值爲"$<"，意思是從鍵盤（正確一點說，應該說是標準輸入，standard input）讀到的東西。接著的"while"你應該清楚，不再贅述。

第 9 行是判斷"變數 cnt 的值是不是被 10 整除？"，如果是，則問"要不要繼續"，如果不繼續（由標準輸入讀到的東西是"n"），就"exit"，跳出殼外，結束；否則，還是繼續數下去，一直到 100 爲止。

建議你將這些 script 輸入你的電腦，跑跑看，自己加些有的沒的，試試看你是不是已經都清楚它們的每個步驟[4]。

再提醒各位，這裡講的"執行"一個 C Shell Script 和第五章講的用"source"去"執行"一個 C Shell Script，很容易混淆。其實它們眞的很像，不同的是，用"source"的時候，並不會去挖一個子殼再執行東西，它保持在同一個殼內做動作；而用 script 本身去執行時，根據 script 的第一行做動作，所以如果第一行是"#!/bin/csh"時，它會"先執行 /bin/csh"（相當於去挖一個子殼），再執行 script 裡面的東西。當然，如果第一行不是"#!/bin/csh"，那這個 script 跟 C Shell 就沒關係，也無所謂的子殼了。

C Shell script 的寫法很粗淺地介紹到此，目的只是要大家不要怕寫 shell script，沒啥好怕的，充其量只是堆砌一些 *UNIX* 指令，以及關鍵字像"foreach"之流，其中沒什麼奧秘之處，如果你還想更進一步學習更多 C Shell Programming 的技巧和方法，在最後一章列的參考書是不錯的範本。

[4] 不要忘了"-v"選項的使用

For millions of years,
mankind lived just like animals.
Then something happened which unleashed
the power of our imagination.
We learn to talk.

from The Division Bell
(Pink Floyd)
by Stephen Hawking

Chapter 8

X Window System

喔，不！ X 視窗跟 *UNIX* 沒有絕對的關係。只是，現在的 *UNIX* 系統很少不具備 X 視窗。在這個時代，連 MS-DOS 都有 MS-WIN 可以用了，具備作業系統大老身份的 *UNIX* 怎會落於人後？下面的介紹，仍然基於使用者的實用角度，不談任何原理及技術，一來，那是本文的目的，二來，筆者本人也不是很懂就是了 :-)。X 視窗一躍成爲近年來電腦，尤其是工作站，不可或缺的工具，主要的原因還在於它實現了長久以來人們使用電腦的基本要求：親切的人機介面。透過 X 視窗，使用者可以清楚的在螢幕上看到電腦的另一個面目，而且不必再局限於鍵盤，將一部分動作交給視窗處理。X 視窗輕易的獲得認可並移植到各式各樣的機台上，不只是 *UNIX* 系統，其他的作業系統也幾乎都可以跑 X 視窗，連 PC 也不例外，喔，不，這裡不是指 Micro$oft Windows，它不算是 X 視窗！（它勉強算半個視窗。）

X Window 處理整個機台前景的運作，其地位自不待言，它的重要性就像 C Shell 處理你的指令一樣，所以它也有一些檔案來規範整個 X 視窗，就像 C Shell 有".cshrc"來規範一樣。這些檔案也可以由你設定，用來控制整個 X 視窗的運作，當然，你也可以不去更改這些檔案、設定，跟"殼"一樣，系統會有預定的值，如果你滿足於這些系統預定值，你就可以不必管，但是在多數的情況下，我們總是希望依自己的喜好和需要來重新設定一些東西。

哪些檔案（啓始檔）來控制 X 視窗呢？最基本的是".xinitrc"和 ".Xdefaults"，另外還有一些檔案在待會兒講到的時候再順便提。下面爲了說明具體起見，以 Sun Sparc 工作站爲例，介紹整個 X 視窗的啓動和設定。由於 X Window 牽涉極廣，這裡只是很簡單的說明，勢必無法滿足很多使用者，但不要忘了，你現在讀的是 *UNIX* 的使用指南，不是 X WINDOW 的使用指南!

開始吧...

8.1 Open the Window

先從啓動說起。

login 之後，我們就進入 *UNIX* 的Shell，所在的這個殼是最外層，也叫"console"，在這裡你會感到很不方便，因爲你只有一個螢幕。Sun 在採用 X 視窗之前，自己也發展了一套視窗系統，叫做 SunView，後來才改成 OpenWindows。OpenWindows 也是 X 視窗的一種，雖然他們之間還有一些差異，但是基本上我們可把它當成一般的 X 視窗，不會有任何影響。

打 "openwin" 其實是執行這個 Bourne Shell "程式" (Shell Script) : "/usr/openwin/bin/openwin"。它會去看你的 home directory 有沒有".xinitrc" 和."Xdefaults" 這兩個檔，有就用，沒有就幫你從"/usr/openwin/lib"裡面拷貝一份樣板檔給你。換言之，如果你進 Window 的方式是打"openwin"，那你其實不必先擁有這兩個檔[1]。

好，現在假設你打 openwin 要進 X Window，系統先根據你的環境參數 PATH 的路徑依序去找一個叫"openwin"的執行檔，正常情況下會在"/usr/openwin/bin"下找到。找到後就去執行它。這個檔案是執行檔嗎？ 怎麼可以用 more，cat 看得到內容？而且不是亂碼哩！一點也沒有執行檔的樣子。這要說到 *UNIX* 的另一個特色：檔案型態(file permission)。*UNIX* 很聽話，你只要說是執行檔，它就是執行檔。你可以把你的任何一個檔案"變成"執行檔！只要 chmod 這個指令就可以了[2]。一旦你"說"某個檔案是執行檔後，你打檔案名稱去執行它時，*UNIX* 就盡一切努力要去執行它。而在 *UNIX* 的傳統裡，如果執行檔不是 binary code 的話，通常會給 *UNIX* 一個"提示"，告訴 *UNIX* 這個執行檔是哪個語言或哪個東東。剛剛說的"/usr/openwin/bin/openwin"就是個例子。它的第一行是：

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

這是告訴 *UNIX* ，"以下的文字是要用 /bin/sh 這個程式來解讀的，他們是 Bourne Shell Script"。也就是說，這些文字全是 Shell Language。所以對於熟悉 Shell 語言的人來說，這些文字就不是天書，但對不懂這種語言的人來講，就是有看沒有懂了。你可能會看到其他一些所謂"執行檔"，是以下面這些文字來開頭的：

```
#!/bin/csh
     或
#!/usr/local/bin/perl
     或
#!/bin/awk
   ........
```

就像剛剛說的，語言不同罷了！CSH、PERL、AWK，都是某一種電腦語言而已。言歸正傳，剛才說到"/usr/openwin/bin/openwin"。它是Bourne Shell Script，它先幫你設定一些重要的環境參數，然後，看一看你"家"有沒有".xinitrc"與".Xdefaults"，沒有的話就拷貝一份樣板給你。最後，眞正執行的是一個叫"xinit"的 binary 執行檔[3]。換句話說，"xinit" 才是 X Window 的主角，"openwin" 這個字是"唬人"的！所以剛剛說：不必敲"openwin"也可以進 X Window！ 現在就來看一看。

你已經有了".xinitrc"和".Xdefaults"，對不對？好，剛剛已經講到"xinit"要執行了，它會去看你家裡的".xinitrc"檔，（跟 C Shell 要跑起來的時候一樣，csh <=>.cshrc ，xinit <=> .xinitrc）根據".xinitrc"來決定你的 X Window 的環境。它先看你家有沒有".Xdefaults"檔，有的話就執行：

```
xrdb ~/.Xdefaults
```

否則就執行系統的樣板：

```
xrdb /usr/openwin/lib/Xdefaults
```

這個指令的意思跟"source ~/.cshrc"是類似的。記得前面講，進 C Shell 時，系統會先根據你的".cshrc" 來設定 C Shell 環境，用的方法就是 "source .cshrc"。現在要進 X Window，系統也是先根據你的".xinitrc"來設定 X Window 環境，只不過用的是另一個指令叫 "xrdb"，它的地位相當於 source ，都是用來"讀取啓始檔，設定該設定的環境參數"。所以可以想見，".Xdefaults"

[1] 這裡埋下一個伏筆，進 Window 的方式，可以不用 openwin！稍後再說。
[2] 參考第二章有關 chmod 的說明。
[3] xinit 在哪裡呢？自己用 which 找一找吧。

也是在進 X Window 時看一下而已，進了Window 就不管了。如果你已經在 X Window 裡面而修改了".Xdefaults"，比如說把 Window 顏色改了，這個新的修正值並不會發生作用，因爲 X Window 根本沒看到！怎麼叫它去看呢？就是用"xrdb ～/.Xdefaults"這道指令！

xinit 根據".Xdefaults"設定好 X Window 的環境後，就要眞正進 X Window 了，在進去之前，請各位想一想，Window 是很有彈性的，你可以在螢幕上開很多很多小 Window，分別做不同的工作，整個螢幕就像一個大桌面，這些小 Window 散佈其間，不管一管的話會亂成一團。所以啦，就要有"管理員"的存在。xinit 根據".Xdefaults"設定好 X Window 的環境後就接著執行"視窗管理員" (Window Manager)，以免等一下進了 Window 桌面亂七八糟。所以，在".xinitrc"裡面，緊跟著 "xrdb"的，就是"視窗管理員"，它也是一個 binary 執行檔，而且有好多種，下面慢慢說。

如果你看看你的"～/.xinitrc"的話，你應該會看到有 "`olwm &`" (或是 "`olwm -3 &`") 的一行，"olwm"就是"視窗管理員"(Window Manager)，它的全名叫做 "Open Look Window Manager"，取第一個字母連起來，即是"olwm"。它是 Sun 發展／出品的一個視窗管理員，因爲具有"drag and drop"的特異功能而廣受歡迎。所謂"drag and drop"，舉個例子，你若想殺掉某個檔案，你只要把滑鼠移到那個檔，點著它，把它拉到垃圾桶的位置，放開，就可以了，那個檔案就被"remove"掉了，像 Macintosh 一樣！而這個功能的啓動，是靠 /usr/openwin/lib/openwin-sys 來啓動的，所以你會看到在".xinitrc"檔裡面，有 /usr/openwin/lib/openwin-sys 這一行在視窗管理員 olwm 前面。

執行視窗管理員之後，就進入 X Window 了，這時，系統會再看看你有沒有要它先幫你擺一擺桌面或先跑一跑甚麼 program ，".openwin-init"這個檔，就是做這個用途的。如果你有這個檔的話，那裡面列的程式就會在你一進入 Window 的時候一一執行。比如，擺個時鐘在右上角，放個信箱在左下角，打開一個編輯器(texteditor)在正中央，等等。你只要在".openwin-init"裡面寫上你要 run 的 program，就可以了。如果你沒有".openwin-init"這個檔，系統就會 run /usr/openwin/lib/openwin-init，幫你把一些基本的tool：cmdtool, file manager, online help viewer，放在螢幕上給你使用。如果你是新手，你就可以利用 helpViewer 來學一學 OpenWindows 的基本操作了。另外，有個 ".openwin-menu"是來設定你的 main menu 的，就是你用 mouse 右鍵在 openwin 裡面拉出來的 menu，它有一定的格式，有興趣可以自己研究研究。

8.2 X Resources

到此，進視窗的程序已經結束。在前面，對 .Xdefaults 這個檔談得並不多，現在來多看幾眼。大家進 X Window後，都會開一大堆視窗，一個接一個。有沒有想過，它們的顏色、長寬比例、裡面的字體大小、字型、視窗開出來的位置，等等特徵是靠甚麼決定的？ 如果覺得 default 的字體太小了，該怎麼改大一點？這些問題，都是有關所謂 "X resource"。每一個 X Window 的program 都或多或少會有一些對應的 X resource，他們用來控制該 program 的外在行爲表現。透過他們，使用者可以有限度但彈性的塑造自己使用的 X program 的門面。例如說，你想修改 OpenWindows 的字體大小，那你可以在你的".Xdefaults"檔裡面加上一行：

```
Window.Scale:  large
```

就可以把字體變大一點(default 值是 medium，12 pixels，large 是 14 pixels)。這個"Window.Scale"就叫做"Resource Name"。每個 X 程式所附帶的 X Resource 都不一樣，大體上，程式的供應者都會提供該程式的 X Resource 清單，讓 user 可以有所依據。也因爲每個 X program 的功能不一樣，實在不可能有一個通用的 X Resource 表列，下面是筆者所用的".Xdefaults"檔的一小部份，不是筆者自己寫的，都是在安裝程式時根據程式說明加(copy)的，初學者大可不必擔心它的複雜度！

```
! X-Windows defaults file.
! olwm.ColorFocusLocked:  True
! For Gremlin to work in any Window Manager
!sx.focus: on
Xarchie*xarchieFont:    9x15
Xarchie*xarchieBoldFont:        9x15bold
!-------------------------------------------------------
! Original Cxterm Input method with NewFace Modification
cxterm*pointerShape: left_ptr
cxterm*VT100.Translations: \
#override       <KeyPress> F1:  string("^[OP")          \n\
                <KeyPress> F2:  string("^[OQ")          \n\
<KeyPress> F3:  switch-HZ-mode(UserCZ-b5) \n\
                <KeyPress> F4: switch-HZ-mode(ASCII) \n\
                <KeyPress> F5:  switch-HZ-mode(PY-b5)   \n\
                <KeyPress> F7:  switch-HZ-mode(ZOZY)    \n\
          Shift <KeyPress> F8:  switch-HZ-mode(QJ-b5)   \n\
                <KeyPress> F9:  switch-HZ-mode(CangJie)   \n\
                <KeyPress> F10: switch-HZ-mode(Punct-b5)    \n\
!~Meta <KeyPress> Escape:  insert() switch-HZ-mode(ASCII)\n
!-------------------------------------------------------
Ghostscript*geometry: 500x500-0+0
Ghostscript*xResolution: 60
Ghostscript*yResolution: 60
! xterm window defaults.
xterm*background: darkslategrey
xterm*foreground: white
xterm*cursorColor: white
xterm*pointerColor: red
xterm*border: SkyBlue
xterm*sunFunctionKeys: True
xterm*jumpScroll: True
xterm*scrollKey: True
xterm*curses: True
xterm*loginShell: False
xterm*font:     -adobe-courier-medium-r-normal--18-180-75-75-m-110-iso8859-1
xterm*scrollBar: True
xterm*saveLines: 600
xterm*iconImage: ~/BITMAP/roc.bitmap
Scrollbar.JumpCursor: True
OpenWindows.MultiClickTimeout: 4
OpenWindows.Beep: always
OpenWindows.SetInput: followmouse
OpenWindows.ScrollbarPlacement: left
OpenWindows.PopupJumpCursor: True
OpenWindows.WorkspaceColor: #40a0c0
OpenWindows.IconLocation: top
OpenWindows.SelectDisplaysMenu: True
OpenWindows.WindowColor: #cccccc
OpenWindows.DragRightDistance: 100
Text.AutoIndent: True
Text.MaxDocumentSize: 40000
Text.LineBreak: Wrap_word
!Text.Checkpointfrequency: 700
```

```
Keyboard.DeleteWord: 17
window.Scale: large
```

8.3 A Little More

剛才好像有說：進 Window 其實不必用 "openwin"這個指令。對，你只要打"xinit" 就可以了。就像前面說的，"xinit"才是主角，"openwin"這個指令只是幫你多設定一些東西而已，你當然可以婉拒它的好意，一切自己來。你只要打"xinit"，視窗一樣會出現。當然，xinit 也會看你的".xinitrc"。那如果你沒有".xinitrc"這個檔呢？Surprise！視窗竟然也會出來。只是門面有一點奇怪，不太甘願的樣子，只出現一個 xterm 而已，甚麼都沒有，連 mouse 都不能用。對了，那是沒有視窗管理員(Window Manager)的緣故，這時你還可以亡羊補牢，在 xterm 裡面下個 "olwm &" 指令給它，把視窗管理員叫出來維持一下秩序就 OK 了！當然，你也可以不用 "olwm"，"mwm" (Motif Window Manager)，"twm" (Tom's Window Manager)，"olvwm" (Open Look Virtual Window Manager)，也都可以用，隨你高興！甚至已經有某個 Window Manager 在 run 了，你也可以把它殺掉(kill process)，再跑另一個 Window Manager，百無禁忌！

X Window 允許你把在某一部機器跑的程式的視窗 display 到另一部去。例如你由機器A login 進系統，然後你開了個 xterm，由那裡面再 rlogin 到機器B跑 matlab，你在的機器是A，所以你當然想把 matlab 畫出來的圖形 display 在機器A上而不是機器B，這時你要做的動作是：通知這兩部機器，互相合作一下，一個負責跑程式，並把視窗 display 到其他地方，一個負責接受別人的視窗。

在機器A（接受別人的window）：
下指令 "xhost + 機器B的hostname"

這是通知機器A：把 X 的門打開，B要進來display它的視窗。偷懶的人
也可以只打"xhost +"，表示來者不拒，接受所有開過來的window，
不管是不是B的。

這個命令每次login只要下過一次就好了，不必每次要跑程式都下。
除非你又要接受機器C的視窗，才要再 "xhost + 機器C的hostname"。

在機器B（眞正跑程式的，把視窗丟給別人的）：
下指令 "setenv DISPLAY 機器A的hostname:0"

這是通知機器B：把 X 視窗丟到A去。
如果你沒有xhost或沒設DISPLAY，你可能會得到像：

```
Xlib:  connection to "machineB:0.0" refused by server
Xlib:  Client is not authorized to connect to Server
Error: Can't open display: machineB:0
```

的訊息。

各位注意到："setenv"設定環境參數"DISPLAY"，所以是對一個 Shell 的作用。試想：你在A開了幾個 xterm (或 cmdtool, shelltool) 又 rlogin (或telnet)到B去，在其中一個你要跑 matlab 而且把 matlab 的 window display 到A來，那你必須下"setenv DISPLAY 機器A的hostname:0" 這個指令在你跑 matlab 的那個 xterm，對於在其他你也連到機器B的 xterm 裡面，剛剛在跑 matlab 的那個 xterm 中下的"setenv DISPLAY 機器A的hostname:0" 是不起作用的。爲什麼？因爲"DISPLAY"是一個 Shell 的環境參數，你在哪個 Shell 下指令，就只對那個 Shell 有作用！你

開的每個 xterm ， cmdtool ， shelltool 都是獨立的 Shell Window，互不影響。換句話說，你即使開了 10 個 Window 都連到同一部機器，他們事實上是獨立的。了解這一點，也就知道爲什麼一定要在跑程式的那個 Shell Window 裡面下"setenv DISPLAY 機器A的hostname:0"了。上面提到的hostname 也可以用該機器的 IP address 代替。你如果不小心，也許會把 Window 開到美國，甚至北極去，只要那裡的機器也偷懶，只用 "xhost +" 將機器的 X Server 開放出來，沒給特定的 hostname。在這裡要特別提一下 X Window 裡面 Server/Client 的區別，前面說的機器A，將某個視窗丟到機器B，在B的螢幕上顯示出來，雖然A是眞正執行程式的機器，但A是 X Client(要求服務)，而B是 X Server(提供服務)，這一點很容易搞混，請注意。

有需要的話，各位可以看看 X Window 的書，這裡就不多談了。如果不想花錢，有一本免費的 X Window Guide 可以在 ftp.tem.nctu.edu.tw:/Chinese/X-UserGuide，或是在 ftp.nctu.edu.tw:/Chinese/ifcss/software/x-win/x-win-guide.b5 找到，自己去拿來看看也很有幫助。

Question: How many Microsoft programmers
does it take to change a light bulb ?
Answer: None.
The company just changes the standard to darkness.

from Newsweek *July, 11, 1994*
by Andrew Schulman

Chapter 9

先別急著走開

9.1 引介一些有名的程式

這裡列出一些比較常見、好用的軟體，它們都可以從網路上免費取得[1]，這裡也只大概介紹它們的主要用途，它們的使用方法請各位自己拿來玩一玩，或問問同學、朋友。大部分的程式都附有manpage，有的甚至有完整的使用手册，如果你的系統有安裝該程式的話，不要忘了用"man 指令名稱"來得到說明。

TEX

LATEX TEX，一言難盡。史丹佛大學的 Donald Knuth 是知名的數學家及電腦科學家，他準備出一系列的"The Art of Computer Programming"這套書，三册出完，當他正在進行第四册的寫作時，出版社拿來第二册的第二版給他過目，結果令他大失所望，因爲印刷科技的進步並沒有令他的書更好看，反而變糟了，尤其是在數學式子和字體上面的缺陷更令他無法接受。所以他就自己寫了一個電腦排版系統，這就是 TEX 的由來[2]。

它對於文件格式，特別是複雜的數學式子等的排版有獨到的優點，也廣被採用。遠自1977 年開始發展的 TEX 到現在已經非常成熟了，而且被移植到無數的機器上面，有很多變形，LATEX 是其中一個，PC 上也有，像 EmTEX 就是。很多人用 TEX 來做 Documentation，有些研討會甚至也接受 TEX 格式的論文，它的輸出檔案是 DVI (DeVice Independent) format。你若看到有以".tex"或".dvi"結尾的檔案，八成就是 TEX 做的。要注意的是，TEX 不是文書編輯器，你必須先編輯好文件內容，再給 TEX 來處理。也就是說它並不是像一般的幕前排版系統所標榜的 WYSIWYG (What You See is What You Get)，至於其中的優劣就不是這裡討論的範圍。LATEX 是 TEX 的巨集，由於 TEX 的使用指令太繁瑣，Leslie Lamport 就把它整理成比較容易駕馭的方式，便成了LATEX。本書就是用 LATEX 排版的。

xdvi 這是用來看 DVI 檔的程式。文件經過 TEX 處理以後，產生的檔案格式叫"DVI"檔。你可以用"xdvi"[3] 在 X Window 下先預覽（preview），然後再決定接下來的

[1] 如果你還不會用 archie、ftp 等程式，請你請教其他人。有關這方面的說明，筆者已經自本書刪除，如果只須要簡單的說明，可找 2.4 版來看。

[2] 根據 Knuth 教授自己的說法："..... we wanted to produce documents that were not just nice, but actually the best ..."。

[3] 或 xdvik。

工作（列印？或是再修改？等等）

dvips 它用來將 DVI 檔轉成 PS 檔案[4]。

texi2dvi 它用來將 Texi 檔轉成 dvi 檔案。Texi 是 GNU 系列處理文件的格式，相關資料可以找 texinfo 這個 GNU 程式來看看。

xlatex 將 TEX 的"編輯/圖形繪製/xdvi/dvips/列印"功能都整合在一起的一個 X Window 程式。

xv 一個廣爲人知的 X Window 程式，可以用來看 TIFF、GIF、JPG，等各種影像檔案。也可以用來做各種不同 format 的轉換，例如，GIF 轉 PS，或 JPG 轉 GIF 等等。

xautolock 這個程式讓可以在你的 X Window idle 一段時間後，自動鎖住你的螢幕和鍵盤，時間可由你設定(default 是 5 分鐘）。

xnlock xnlock 是另一個 lock screen 程式，lock screen 後會有一個大鼻子小矮人在螢幕上散步並顯示一些你設定的 message。和前一個程式合用頗爲方便：`"xautolock -locker xnlock"`。

xfishtank 它會讓你的 X Window 的背景（Background）變成一個大魚缸，有一些魚游來游去。

xroach 這是個惡作劇的程式，它會在 X Window 的背景放幾隻蟑螂，蟑螂會躲進每個視窗後面，伺機出遊！

xantfarm 讓你的視窗背景長螞蟻，這些螞蟻會慢慢把你的視窗背景吃成蟻丘！

Ghostscript Ghostscript(gs) 是類似 Postscript(PS) 的程式語言。"gs"這個程式可以讓你在沒有 Postscript Printer 的情況下也能列印 PS 檔案。另外可以當作 PS 檔的預覽程式，也可以用來將 PS 檔轉成其他格式，例如 GIF、TIFF 等影像檔，是筆者看過唯一可以把 PS 轉成其它 image 檔的免費軟體:

```
% gs -sDEVICE=gif8 -sOutputFile=tiger.gif tiger.ps
```

ghostview Postscript file 的 previewer，利用到"gs"做 Postscript Language 的翻譯器。

xfig X Window 上的畫圖軟體。類似的軟體還有 xgremlin、tgif 等等。

xvgr 作圖軟體，可畫各種曲線，對數據資料的處理很有用，類似下面會講到的 gnuplot。

groff GNU 的 document formatting system，熟悉 *UNIX* troff 及 nroff 的人可試試。筆者常用它來將 manpage 轉成 PS 檔，例如:

```
% groff -man -Tps /usr/man/man1/cat.1 > cat.man.ps
```

elvis,vim 另一個跟 vi 相容的文書編輯器。

[4] 或是"dvipsk"。

celvis 中文 vi，在 celvis 裡面你可以輸入中文。

gcc/g++ GNU C/C++ Compiler！

ispell 拼字檢查軟體。

gzip GNU Zip！一個廣受使用的壓縮程式，其輸出檔爲 ".gz"（*UNIX* 的標準壓縮程式是"compress"，解壓縮程式是"uncompress"，壓縮檔爲".Z"結尾）。gzip 可以壓縮(`gzip file`)、解壓縮（`gzip -d xxx.gz`）。

gnuplot Plotting 軟體。很好用，可做 3 度空間的圖，對研究資料的整理很有助益。

gnufit gnufit 是 gnuplot 的擴充版，可以做 curve fitting！

PSUtil 包含多個 PS 檔案處理程式的軟體，諸如：PS檔的分頁、再處理。用來處理 PS 檔案極爲好用。筆者最常用其中的"psselect"來將一個 PS 檔分成奇、偶數頁做雙面列印，以節省紙張、方便裝訂。

psdraft 一個小小的 PS 檔案再處理程式，它可以把一個 PS file 的每一頁再印上"DO NOT COPY"或其它你自己要印的字在背景上。

pscount 計算一個 PS file 的總頁數，利用到"gs"。其實它只是一個小 Shell Script：

```
#!/bin/sh
# Count number of pages output from a postscript file by using
# ghostscript (2.6.1pl4) specific device properties.
# - Kevin Grover, grover@isri.unlv.edu, 9 Mar 1994
#
# to redefine ^D, use "(\004) cvn {} def"
#
(
cat $*
echo currentdevice /PageCount gsgetdeviceprop == flush
) | gs -q -sDEVICE=bit -sOutputFile=/dev/null -r5
- | tail -1
```

cxterm 中文 "xterm"，目前最廣被使用的中文視窗。

cnprint 列印中文的程式，可接受 Big5 和 GB 兩種格式的中文碼。也可以把中文檔轉成 PS 檔，例如:

```
% cnprint -5W -f=k48 BIG5_file file.ps
```

mosaic 一個用來與 WWW server 連線的 client 程式。近來，另一個叫"netscape"的程式也漸流行。

elm 這是近來被廣爲使用的電子郵件軟體，比傳統的 mail 程式更好上手，如果你受不了傳統的 mail 程式，試試 elm 吧！

pine 這也是另一個電子郵件軟體，比 elm 更簡單，初學者都喜歡。

ftptool 架在 X Window 上，視窗模式操作的 ftp 軟體。

perl	一個綜合 Shell，awk，sed 的新 *UNIX* 語言，由 Larry Wall 所創。他也是補丁程式"patch"及讀 news 程式"rn"的作者。
nenscript	將一個 ASCII 檔案轉成 PS 檔的程式。另有一個叫"a2p"的程式也做同樣事。
bvi	Binary vi。一個可直接編輯 binary 檔案的編輯器，操作方法和 vi 類似。
mpeg2play	解壓縮並播放 mpeg 檔案的軟體。

接著列出一些在 *UNIX* 上常看到的檔案名稱，這些是約定俗成的名字，沒有絕對的強制性。

.ps	Postscript 檔案。可用"gs"來看，也可以由接受 postscript 的印表機直接印出來。
.eps	Encapsulated Postscript 檔案。Postscript 檔案的一種。
.gz/.z	GNUZIP 壓縮過的檔案。可用 gzip (gnuzip) 來解。
.zip	ZIP 檔案。可用 unzip 這個程式來解。
.zoo	用 zoo 這個程式來解。
.shar	Shell Archive。用 unshar 來解，或是自己用 /bin/sh 慢慢解。
.tar	*UNIX* 包裹。用 tar 來解。
.hqx	Macintosh 檔案，用 binhex 來解。
.tgz	通常是 .tar.gz 的縮寫，爲使附檔名在 3 個字，所以叫 tgz。先用 gzip 再用 tar 解開。

9.2 參考一下

以下列了一些參考書／文件，有興趣的人可以找來看一看。這其中，有些是筆者寫作本書的參考資料，特別在此聲明以避免不必要的誤會。

UNIX C Shell Field Guide

By Gail Anderson & Paul Anderson/Prentice-Hall

這是講 C Shell Programming 的書，寫得很不錯，市面上也有中文版。書裡面教你一些 C Shell的指令、用法及如何自己寫 C Shell program。想精通 C Shell 的人不可錯過這本書，號稱 The Bible (of C Shell)！

Zen and the Art of the Internet

A Beginner's Guide to the Internet, First Edition, By Brendan P. Kehoe

這本書可由網路上免費拿到，裡面的內容包羅萬象，任何在 Internet 上會碰到的，這本書都有明確的交代，是"網路行者"們的入門書，只有 96 頁，你可以很輕鬆的躺著讀完。你可以用 ftp 在 leica.ccu.edu.tw:/pub/internet/docs/zen-1.0/zen 下取得。另外，1994 年 8 月 8 日的"Newsweek"有一篇名爲"The Birth of the Internet"的文章，詳細介紹了"Internet"的誕生過程，對"Internet"有興趣的人可以找來一讀。

Big Dummy's Guide to the Internet

A round trip through Global Networks, Life in Cyberspace, and Everything..., (by EFF)

如果說上一本書只是開胃菜，無法滿足你的大胃口的話，試試這一本厚達 250 頁的手册吧！網路人絶對不可缺少的大寶典之一。雖然是以美國的使用者爲對象，這本書仍不失其普遍性。如果說"Zen and the Art of the Internet"開胃的話，這本書絶對足夠讓你撐死，不信？去抓回來印吧！(以上兩份文件可由 ftp.eff.org:/pub/Net_info/Guidebooks 取得)

Internet實務手册 by 曾瑞源

不想讀英文?試試這一本曾瑞源先生所寫的"Internet實務手册"。可以在書店找到，或從交大拿(ftp.nctu.edu.tw:/Chinese/YuanInfo)

The Cuckoo's Egg

Tracking a Spy through the Maze of Computer Espionage, By Clifford Stoll / Doubleday

一個眞實故事。作者 Cliff Stoll 由電腦會計帳目(Accounting)上 75 分錢的不明來源，進而追出一件跨國的間諜案。在本書中你將看到一個身在德國的年輕人如何透過通訊網路，闖入美國大學、國家機構、包括軍事基地的詳細過程。你也將看到一個天文學家如何鍥而不捨，找出事情眞相的精采過程，其中甚至牽涉到 FBI，CIA 以及 KGB。對電腦網路及 Computer Security 有興趣的人可以看一看，保證精采！如果你是一個偵探迷，本書更不可錯過[5] ！

The Worm Story

Communications of the ACM, June 1989, Vol. 32, number 6

想知道一個電腦安全專家的兒子如何放出它的小程式到 Internet 上，導致無數機器當機嗎？一隻會自我複製、自我旅行的"蟲"藉著 sendmail 的 bug 自己散布到 Internet 上的其他機器，糟的是，Robert Tappan Morris, Jr. 這個康乃爾大學的研究生、這隻蟲的創造者自己也無法再控制它。諷刺的是，他的父親，Robert Morris, Sr. 卻是國家安全局的頂尖電腦科學家、一個國際馳名的電腦安全專家...

這不是科幻小說，是幾年前轟動全世界的眞實事件！

Cyberpunk (電腦叛客)

[5] 這本書台灣還沒出現中譯本，甚爲可惜。

中文版已由天下雜誌出版社出版(ps. 非廣告)，報導三件網路史上最出名的"犯罪"事件，當然也包括前面提到的蟲(The Worm)，以及 Cliff Stoll 抓到的來自西德漢諾威的黑客(Hannover Hacker)。看看這本書也許可以讓你了解到網路安全的重要性。書中主角之一的 Kevin Mitnick 在今年初 (1995年二月) 又被逮了，美國時報週刊的標題是這樣形容這位 31 歲的年輕人的："A cyberthief who couldn't stop himself"。詳細內容可參考 1995 年 2 月 27 日的美國時報週刊(Newsweek)。

Life With Unix/A Guide for Everyone

By Don Libes & Sandy Ressler / Prentice Hall

這不是一本 *UNIX* 的入門書籍！但，這是一本可以躺著看，看完之後意猶未盡的書，內容精采容易消化，你所從來沒聽過的有關 *UNIX* 的軼聞、傳說、糗事，在這本書中可以得到答案。*UNIX* 新手、老手都不會嫌棄的一本書。值得一讀。

The *UNIX* -HATERS Handbook

By Simson Garfinkel, Daniel Weise & Steven Strassmann / IDG Books

學了、用了 *UNIX*，結果還是對它很感冒，怎麼辦？ 沒關係，你不是唯一的一個，而且有人還費心寫了一本書來數落 *UNIX* 的種種不是。

不論你對 *UNIX* 是喜歡還是厭惡，看看這本書吧，愈是了解 *UNIX*，就愈能享受閱讀這本書的樂趣。 Donald A. Norman 在這本書的前言說得好：

> I remain suspicious: would anyone have spent this much time and effort writing about how much they hated Unix if they didn't secretly love it? I'll leave that to the readers to judge, but in the end, it really doesn't matter: If this book doesn't kill Unix, nothing will.

嗯...至少 *UNIX* 還值得人家去"討厭"，有些 OS 還沒這份榮幸呢！

The GAWK Manual

想學"awk"？不想花錢買書？自己印一本吧！這是由 Richard Stallman 親自參與編纂的使用手冊，雖然是針對 GNU AWK 所寫，但仍是初學者的必備入門書。趕快到網路上抓回來吧！(任何 GNU 的 ftp site 應該都有，例如 ftp.nctu.edu.tw:/*UNIX* /gnu)

GNU Manifesto

要瞭解 GNU 及 Richard Stallman 的基本理念，最好的方法就是去拿 GNU 自己發表的說明文件。它們可以在任何存放 GNU 軟體的 ftp site 找到，還有一篇 Richard Stallman 接受 BYTE 雜誌的專訪內容，值得一讀。另外，熱訊雜誌 1991 年 12 月號有一篇"Stallman 和他的免費軟體世界" 對 GNU 有不少的介紹，對本書第三章的寫成，有很大的幫助。

如果可以，用 mosaic 到 http://www.cs.pdx.edu/~trent/gnu/ 看看有關 GNU 的文件。

網路

電腦網路是現今世界上最經濟、最快速、最大的資源，懂得利用網路可以讓你以最快速度跟上世界潮流，學會你想學的東西。在網路上， BBS 也好， Net News 也罷，都有討論 *UNIX* 的頻道，常上去瞧瞧可以讓你開開眼界，輕輕鬆鬆學會更高深的 *UNIX* 技巧。

另外，在 ftp.csie.nctu.edu.tw:/pub/CSIE/contrib/cfaq/unix 有中譯的 *UNIX* FAQ，建議大家在上網路發問之前，先讀過它，看看是不是可以從中得到解答。同樣目錄下的 README 檔也列出了一些可以自由取得的相關電子文件，也建議大家先看過一遍。

關於在 *UNIX* 上使用中文的問題，可以在 ftp.csie.nctu.edu.tw:/pub/Chinese/chinese-text/big-faq 中得到部分解答。

9.3 背景說明

本書寫作的相關背景說明:

◆ 中文視窗	X11R5 + CXTERM 5.0 版
◆ 中文輸入	CXTERM 拼音輸入法 + celvis (Chinese VI)
◆ 排版系統	LaTeX2ε + CJK + dvips
◆ 使用機器及作業系統	Sparc10 with SunOS 4.1.3 HP 9000/735 with HP UX/9.05
◆ TeX字型轉換/編譯	TTF2PK

9.4 Finale

作爲一本入門的手册，本書勢必無法滿足大部分 *UNIX* 使用者的需要，讀者們若有其它疑問，應該自行再找相關的書籍參考。

然而，這就是全部了，暫時沒有時間再寫。希望本書多多少少給了一些人一點幫助，如果眞是這樣，那我的目的也就達到了 ...。

It's funny.
Don't ever tell anybody anything.
If you do, you start missing everybody...

from The Catcher in the Rye
by J. D. Salinger

Isn't this where ...