戲說 Unix 馬蓋先

政治大學資科系 連耀南

1. 話說從頭

± - ^

資訊技術日進千里，辛苦所學的技術常常於轉瞬間被後起之秀淘汰，資深技術人員常常因爲忙於專案而無暇練功，以致基本的技術能力，尤其是程式設計能力，常常不如後進的技術人員。資訊專業人員的安身立命之道不在於如夸父追日般苦苦追趕這些移動目標 **(moving target)**，而在於堅實基礎知識及專業領域知識**(domain knowledge)**。利用**C/C++/Java**等程式語言設計一些小工具所耗的時間動輒數小時或數天，除非是一輩子從事程式設計工作，否則資訊人在這種語言上的設計能力將會因疏於練習而隨著時間之推移而漸漸喪失。可是，在數位時代幾乎人人都須處理越來越多的資訊，例如：清除一堆過期的檔案、或一堆佔用空間的影像檔、或將一堆數位相片檔案重新命名，或從大量的實驗資料中抽取有意義的資料等。視窗作業系統固然越來越方便，但也時常需要大量的重複進行某些操作，且不說時間上的浪費，因重複太多機械性動作而罹患腕隧道症候群的資訊從業人員更比比皆是。資訊人很自然的會想到設計一些小程式來幫忙處理重複性工作。可是通常因爲時間極爲有限且程式設計能力生疏以致事與願違。如同拳師一般，如果不能「拳不離手」時時磨練拳技的話，將漸漸的失去矯健的身手。就以資訊相關科系教師而論，因爲終日埋首於教學研究工作而生疏於**C/C++/Java**等程式語言設計能力者，十有八九。有鑑於此，資訊人有必要通曉一些可以隨手使用的程式語言，一方面可以方便快速的協助處理重複性的工作，一方面可以經常隨手磨練。筆者強力推薦**Unix**作業系統所附的各種小型程式設計語言。

Unix 自**70**年代問世以來風行於大大小小各種多人多工的電腦系統，深受系統管理者及玩家之喜愛，非常重要的原因之一是：因爲 **Unix** 具備不錯的軟體工具及環境，可讓使用者很方便的組成各種各樣的工具，可以大幅提高工作效率。如同電視影集裡的馬蓋先，經常利用手邊毫不起眼的工具或物品，運用簡單的物理原理組成非常好用的救命武器或工具。而我們如能善用 **Unix** 提供的環境及基本的工具也能利用簡單的指令組成強大的高階指令，不需用**C/C++/Java**等語言辛辛苦苦的撰寫程式。此種程式設計環境因爲都以 **shell** 作爲最主要的語言，故俗稱爲「**shell programming**」**,** 除了**shell** 之外，常用的程式工具爲 **sed, awk, ex, perl**。而用於產生網頁的工具程式則爲 **PHP.**

筆者使用 **Unix** 凡二十餘年，在電腦上寫論文、處理資料、交換 **Email**、製作**HTML**投影片、改作業、打分數、等等許多繁瑣的工作，均能藉由 **shell script** 強大的能力於彈指之間解決。雖然桌上的電腦仍然是跑微軟視窗，但最常開的軟體卻是 **telent**、**Firefox browser** 及一個類似 **Cygwin** 的 **Unix emulator**。大部分的工作是透過**telnet** 在**Unix server** 上完成的，而處理個人電腦裡的檔案也是經常使用**Unix emulator** 去做的。視窗系統與**Unix** 搭配使用，成爲最佳的工作模式。筆者的教學網頁裡數以萬計繁複的網頁，就是憑著 **vi** 加上 **shell script** 獨力完成的。在這裡，筆者先介紹幾個小例子以展示 **shell script** 的能力，再來談談一些經驗作爲引玉之磚。（請注意：本文所提供的小程式只能當成「私房菜」，做爲個人的私有工具。如欲供大眾使用，則需更爲嚴謹的設計。）

以下的指令將以檔名爲 **Image** 開頭的 **JPG** 影像檔全部更改檔名，在原檔名之前加上**"nccu"**字串。

```
for i in Image*.jpg
do
mv $i nccu$i
done
```

當然，如果嫌檔名太長的話，可以加幾個指令，可以做得更好。

如同一般的程式語言一般，要完成一樣事情並非只有一種方法，但如果使用相同演算邏輯的話，程式本身多數是大同小異的。 **Shell programming** 其實也差不多的，但可用的演算邏輯卻遠非一般程式語言可比。 **Shell** 有很多「巧門」，如能善用這些巧門的話，將可省下許多編撰程式的時間。例如：可以不利用迴圈產生連續數字，如 **1** 至 **1000**，對此任務，一般人很自然的想到用迴圈（**loop**）的方法，使用 **C/C++/Java** 等程式語言撰寫一個程式，經過編譯與執行，得到一支程式。以下是利用迴圈方式所寫的一個 **C** 程式：

```
#include
main(void){
   int i
   for (i=1; i<=1000; i++){
      printf ("%d\n", i);
   {
{
```

這樣的 **C** 程式雖然簡單，若不是程式撰寫員，一般資訊人員也得翻翻書才能記得所有細節。以下是利用迴圈的 **shell script**：

```
count=1
while [ $count -le $1 ]
do
    echo $count
    count=`expr $count + 1`
done
```

如果對**shell** 語法不很熟悉的話，這樣一個**shell script** 也不是三五分鐘可以寫出來的。不過，可別忘了利用巧門。筆者常用的偏方是，從系統中找一個超過**1000**行的檔案（假設檔案名爲 **testfile**），去執行下面這樣的指令：

指令	**head -1000 testfile | cat -n | cut -c1-6**
解釋	**head -1000 testfile** 這個指令將 **testfile** 的前**1000**行抓出來，而 **cat -n** 會把讀到的資料加上行數送到 **STDOUT**，而 **cut -c1-6** 則把前六個 **character**「砍」下來，正好是行數。

如果你常用 **cut** 這個指令的話，這個方法只需構思數分鐘，再花個數秒鐘直接從鍵盤鍵入指令，就可輕易得到結果。如果你熟悉 **awk** 的話，下面這個方法更爲簡單：

```
head -1000 testfile | awk "{print NR}"
```

如果對 **awk** 或特定工具程式不熟也無傷大雅，反正條條大路通羅馬，前面兩種方式也可做到同樣的事。

以上小例子看似簡單，但它可是一種另類的程式，讀者可發揮無限的想像力發掘更多的巧門。對於一般非專職程式設計師的使用者而言，使用 **head**、**cat**、**cut** 等這些常用指令遠比寫一個

C/C++/Java 程式更爲熟練，所花的心力通常微不足道。二十餘年來，筆者經常隨手花個數分鐘時間寫個shell script 幫忙處理瑣事，省下不少時間。由於所費時間不多，通常不需刻意保存所設計的script，絕大部分僅用過一次就丟掉，下次有需要時，重新再寫一次還比花時間去找留存的shell script還更省事，只有時常會用到的script才有留存供重複使用的價值。當然，如果要設計一個大型的程式，規規矩矩的按部就班去撰寫程式還是絕對必要的，

要具備善用 Unix 的功力的話，傳統的程式訓練是不夠的，必須對Unix 的環境及各種工具程式的特性有相當深入的瞭解，尤其是對於用來組合各種工具程式的程式—shell，更需有透徹的瞭解。

2. Unix 之優缺點

+ - ^

任何一種程式語言都有其最適用的情況及其罩門，Unix shell 也難逃此宿命。它有幾項優點:

1. Script 都是純文字檔，不被特定軟體綁住

大部分的 script 檔案都是以文字檔存放，一個程式所產生的檔案很容易爲其他程式使用，一個程式在使用其他程式所產生的檔案時也不需知道是何種程式所產生的。比起現行個人電腦視窗系統上的各種使用專屬格式的檔案系統而言，方便很多。

2. 不需編譯，非常容易在不同系統下執行

shell/awk/sed/ex/perl script 都是純文字檔，也不需編譯即可使用，非常方便，從一個系統移植到另一系統時，直接搬過去即可，雖然可能要更動有些系統設定，但是因爲不需重新編譯，可省下不少麻煩。

3. 適合用於開發簡易小軟體

很多script 長度很短也很簡單，是所謂的 "little program"。當設計一個個人用的 script時，幾乎不需另外寫使用手冊，直接看script 就可以瞭解其用法。要維護也很簡單，不需撰寫複雜的維護手冊或設計手冊。當然，如果是要寫一個供公眾使用的 script 時，這些囉唆的額外工作還是要按部就班去做的。

4. 因開發耗時極短，玩家可隨時根據自己的需求動手開發或更改

如果是寫給個人用的 Script，當然可以隨意的根據自己的需要設計，比起用他人的程式而言，不但無法完全符合個人需求，還須花時間去尋找適用程式，更須花時間去 K 使用手冊。依據筆者的經驗，大部分情況下，寧可自己寫個 shell script 還比較快。

缺點當然是免不了：

1. 執行效率較差

因爲是用 Interpreter 而且是用間接的方式達到目的，其執行效率不佳。例如，前面所舉的數字產生器，如果是用head, cat, cut 去組合成的話，可能比傳統的 C/C++/Java 的迴圈方式慢很多。可是，話說回來，0.001 秒與1秒的反應時間，對終端機前的使用者而言，其實沒什麼分別。可是在程式設計上所花的時間卻是數分鐘與數小時的差別， shell script 的設計可省下數小時甚至數天的時間，那差別可大了。

2. 力有未逮之時

shell 畢竟不是傳統的 **general purpose** 的程式語言，有些事是做不到或很難做的，例如數學運算。所幸，後來發明的 **perl** 程式語言融合了 **shell script** 及傳統程式語言的優點，其能力已經不下於傳統的程式語言。此外，因爲大部分的現成 **script** 程式語言都是以處理文字檔爲主，因此，非常不容易處理多媒體檔案，例如最常用的 **grep** 指令就無法發揮作用。對於非英文的文字檔，也常常出現問題。

3. 漏洞難免

由於設計給個人用的程式不免爲了省時省事而忽略了很多意外的處理，使用不慎時會有嚴重後果，必須小心。話說回來，如果設計用完即丟的小程式時也要將所有意外情況都考慮清楚，那也省不了多少設計時間，也就失去了原有 **little programming** 的立意。有得必有失，得失之間，得由使用者視情況決定。

3. 學習 Unix Shell Programing 之方法

+ - ^

1. 熟悉 **Unix** 上各種基本指令及小型工具程式

市面上有很多書，網路上有很多教學網站可以去挖，筆者的教學網站 **(www.cs.nccu.edu.tw/~lien/UNIX.htm)** 也有一些資料，如今的資訊專業人員可以很輕易的挖出來。有用的小型工具程式 **(little programming language)** 包括 **shell, sed, awk, ex** 等。此外，**perl** 是一個更高級的**programming language,** 很多要利用好幾個程式工具合作才能完成的事，都可以輕易的在**perl**中解決。但是，因爲**perl** 包含了很多 **shell/awk/sed** 的功能，也比較繁瑣，最好等歷練過 **shell/sed/awk/ex** 這些之後才來學 **perl** 會比較容易上手。最後，如果要製作含有 **CGI** 的網頁時，學學**PHP** 等專爲**CGI** 設計的語言，也是很重要的。

2. 盡量熟悉 **Regular Expression**

Regular Expression 是 **shell programming** 之所以非常強大的關鍵之一，在 **awk/sed/ex/vi** 都有相容的 **regular expression**，使用者務必盡早熟習，對於受過正統資科訓練的人而言，熟習**regular expression** 並非難事。

3. 熟悉各種特殊符號及其用法

每一個工具程式都會有一些特殊符號，例如**regular expression**，當一個**script** 是叫用多個工具程式時，這些特殊符號必須使用層層的**escape**符號來配合。 **script** 的內容會變得非常奇怪，難以解讀。使用者必須有能力克服這個困難才能盡情發揮**shell script** 的功能。

4. 熟悉 **Unix** 上各種系統檔案的路徑

例如：**/usr/spool/mail, /usr/bin** 等，至少我們可以加以運用，在我們舉出來的實例中，就有利用到 **spell** 所用的字典檔案來設計自己的**script**的。

5. 熟悉 **Unix** 基本精神

例如 **directory** 及 **device driver** 在**Unix**中均視爲檔案，其管理方式就像管理檔案差不多。

6. 熟悉 **Unix** 程式設計的一些習慣

例如：而所有產生的資料盡量由 **STDOUT** 輸出，以方便組合成其他指令。如果該指令可用來用在 **"|"** 管道之中的話，其資料盡量由 **STDIN** 輸入，話說回來，最合適的方式仍應取決於使用者之需求。

4. Shell Script 設計小秘訣

+ - ^

1. 定期備份，隨時備份要更改的檔案

所有資訊使用者最重要的習慣是定期備份，在**Unix** 下，更要隨時備份。 越方便、能力越強的系統，其破壞力也越強大。例如: 如果某個使用者下一個如下的指令要清除一些檔案：

```
rm -r  foo*.*
```

不幸因爲打字太快，因而多了一個空白：

```
rm -r  foo*. *
```

結果是，這個使用者所有的檔案將被刪除!

2. 慎重選用適當的工具程式

雖然是條條道路通羅馬，但是各種工具程式各有其特性，使用者最好瞭解各種工具程式的特性，再據以挑選適當的工具來用。選用適當的工具會讓你事半功倍，否則可能是自找麻煩。

3. 盡量自動化

只要能自動化就自動化（能偷懶就偷懶），讓**shell** 負責執行繁瑣的重複性工作。

> **Don't use hands to do things that can be done efficiently by the computer. -- Tom Duff**

4. 不要勉強自動化

不要爲自動化而自動化，有時候直接用手去做反而更快。

> **Don't use computer to do things that can be done efficiently by hand. -- Richard Hill**

我們所舉的例子中，就有要求使用者進入**vi** 去做事的，並非全部交由**shell**自動去做。不過，爲了磨練**shell programming** 的技巧，有時候難免多花點時間找點苦頭吃，也是不錯的。

5. 適時搭配 **PC** 視窗系統

如果搭配視窗系統去作比較方便時，那就不要硬在 **Unix** 上作。有時候某些動作在**Unix** 上很難做到時，可以把中間過程所產生的檔案傳到**PC** 上，利用視窗系統上較爲方便的工具處理之後，再上傳回 **Unix** 系統繼續處理。

5. 一些有趣的 Shell Script 實例

+ - ^

以下所舉之範例均省略掉必要的意外處理，使用者在使用時應注意到這個問題，尤其是在更動檔案時，避免對檔案造成意外的更動。

這裡的 **shell script** 分爲兩種，一種是直接在 **command line** 下的，所有參數都是直接給定的，另一種則是存成檔案做成可以重複使用的指令，所需的參數在**script** 中用變數代表（例如 **$1, $2, $***等變數**)**，由使用者在叫用該 **script** 時才給定的。在「用法」裡有說明**Script**的叫用方式，但有些 **script** 沒有標明「用法」，讀者可以從 **script** 中輕易的看出如何使用，自行製作成指令。

5.1 簡單的 script

功能	**grep pattern from several files, output with line numbers**
用法	**gn [options] <pattern> <filename1> <filename2> ...**
Script m1	**grep -n $***
解釋	可以用來檢視哪些檔案含有某一字串，這個 **script** 很單純的將使用者所下的**grep** 指令加上 **-n** 選項而已。讀者如果會常常用到時，建議製作成指令以便重複使用，省點打鍵盤的功夫。

功能	**check the tail lines of your mail box**
Command m2	**tail -100 /usr/spool/mail/uname**
解釋	假設系統將各人的信箱放在 **/usr/spool/mail/yourname** 這個指令將信箱中的最後**1000**行印出來， 可用來檢查是否有新信件進來。
註	新信件如有附加檔或**encoded text**，不容易檢視內容。

功能	**double space a text file**
Command m3	**sed G <filename>**
解釋	**G** 是 **sed** 的指令，將 **buffer** 的內容加在每一行後面 因爲**buffer**內的初始值是空的，其效果等於在每一行文字後面加上一個空行。 請注意，**G** 是 **sed** 內的一個一般性指令，並非特意爲**double space**的功能所設計的。

功能	**count the number of lines for a file, number only**
Command m4	**wc -l $1 | cut -c1-8**
解釋	**wc** 這個指令會列出一個檔案的行數、字數、及字元數，其格式隨系統而不同，使用者必須依據實際情況調整。在很多**script**中都會用到檔案的行數作爲參數，所以有必要設計一個指令將所需要的資訊從**wc** 這個指令所輸出的一堆資訊中提出來。

功能	**generate a sequence of numbers**
用法	**count <number>**
Script m5	**count=1** **while [$count -le $1]** **do** ** echo $count** ** count=`expr $count + 1`** **done**
Script m6	**head -$1 <longfile> | cat -n | cut -c1-6**

解釋 m6	**head -$1 longfile** 這個指令將 **longfile** 的前 **$1** 行抓出來，而 **cat -n** 會把讀到的資料加上行數送到 **STDOUT**，而 **cut -c1-6** 則把前六個字元「砍」下來，正好是行數。 使用者可以自己找一個或製作一個長檔案作爲 **longfile**。
Script m7	**head -$1 <longfile> | awk "{print NR}"**
Script m8	**head -$1 <longfile> | perl -p 'print $.'**

功能	**Instant Message**
Command m9	**echo "Hello!" > /dev/ttyxx**
Command m10	**cat filename > /dev/ttyxx**
解釋 m9,m10	**Command m9** 將**"Hello!"**字串送到終端機 **ttyxx** 上，而 **Command m10** 將檔案 **<filename>** 的內容送到終端機 **ttyxx** 上，這就是一種**instant message**。 終端機在**Unix** 裡被視爲一個檔案，其權限通常是對所有使用者開放的，（當然使用者可以關閉權限），任何使用者可以將任何資訊送進這些檔案，其效果等於將資訊展現在那個終端機上。使用者可以利用 **who** 這個指令，找出收訊者的終端機編號。（**uid** 後面即是 **terminal id,** 將前面加上 **/dev/** 即可得到對應的檔名）。
Command m11	**cat < /dev/ttyxx**
解釋 m11	既然別人的終端機可以寫入，那當然也可以讀，這個指令可以讓人偷窺他人在終端機上所敲的任何鍵**!!** 不過，別慌，系統的初始設定應該會將他人讀這個檔案的權限關閉，只有 **super user** 可以讀得到這個檔案。

功能	**search a file recursively**
Command m12	**find $HOME -name <pattern> -print**
Command m13	**find . -name <pattern> -print**
解釋	**Script m12** 從 **$HOME** 開始往下搜尋名爲 **<pattern>** 的檔案， **Script m13** 從現在的工作目錄開始往下搜尋名爲 **<pattern>** 的檔案。 **<pattern>**裡可以含有萬用字元，但必須包含在""之內，以免被 **shell** 在執行之前即被展開。

功能	如何讓程式在**logout**以後還能繼續執行
Command m14	**nohup command [arguments] &**

功能	**How to make a "script" to judge which the "shell" we are in?**
Command m15	**echo $SHELL**

功能	**finding files less than a day old**

Command m16	**find / -amin +120 ...**
Command m17	**find . -newer <filename> -print**
Command m18	**find . -mmin +120 -type f -exec rm -r {} \;**

功能	分割檔案 **split**
Command m19	**split -b 1m <filename>**
解釋	將一個檔案分成數個 **1mb** 的檔案

功能	**delete file when its size = 0 byte**
Command m20	**find / -size 0 \(-name *.dat -o -name sh*.dat \) -exec rm -f {} \;**

5.2 爲一堆檔案更改檔名

功能	**change file names for a group of files**
Script f1	**for i in Image*.jpg do mv $i nccu$i done**
解釋	這個 **script** 將以檔名爲 **Image** 開頭的 **JPG** 影像檔全部更改檔名，在原檔名之前加上**"nccu"**字串。當然，如果嫌檔名太長的話，可以加幾個指令，可以做得更好。 如果要將數位相機照得的一堆影像檔改個較有意義的檔名時，這非常好用。

功能	**change file names for a group of files**
Script f2	**ls | sed 's/..*/mv & &/' > script vi script sh script**
解釋	這個 **script** 比較**general,** 它是產生如下的編輯指令 **mv <oldfilename> <oldfilename>** 再叫出 **vi** 編輯器讓使用者自己將第二個 **<oldfilename>** 改成新檔名，如此製造一個更改檔名的 **shell script**，最後再叫出 **shell** 來執行這個 **script**。讀者可自行將**vi** 取代爲自己喜歡的編輯器。

功能	將 ***.foo** 改成 ***.bar**
	for i in *.foo

Script f3	do mv $i `basename $i .foo`.bar done

功能	將 foo.* 改成 bar.*
Script f4	for i in foo.* do tailname=`echo $i |  sed -e 's/^foo\.//'` mv $i bar.$tailname done

功能	rename all files in the currect directory to the lower case
Script f5	for file in * do lcfile=`echo $file | tr "[A-Z]" "[a-z]"` mv $file $lcfile done
解釋	如果瞭解 tr 指令的功能，即可瞭解這個 script

功能	sort and rename files
Script f6	for i in 000??.all do sort -r $i > `echo $i \| sed 's/\.all$/.sort/'` done

5.3 轉檔程式

功能	convert text file in DOS format into UNIX format
Script c1	ex $1 <<% 1,\$s/^M\$// w %
解釋	這個指令是用來解決 DOS/Windows .txt 檔案與 Unix 文字檔之間格式不同的問題，在DOS 的 .txt 檔案中，行與行之間用兩個符號作爲間隔，而 Unix 中只用一個間隔符號，因此必須除掉一個。 在script 中，^M 是一個控制字元，請不要誤解成 '^' 及 'M' 兩個字元。 此外，shell 將"<<%" 與 "%" 兩個符號之間的 text 送給 ex 作爲使用者之鍵盤輸入，由於 $ 這個字元在 shell 中是特殊字元，必須加上escape字元，其原始 editing script 如下： 1,$s/^M// w

功能	**filtering out words: is, this, a**
Command c2	**sed "s/this//;s/is//;s/a//;s/ *//g"**
解釋	Input: this is a dumb question Desired Output: dumb question **if you want to filter 'is' but not "isn't", that wouldn't work; you could still use sed, but I think I'd use perl so that I don't have to code 3 sed substitutions for every word:**
Command c3	**sed "s/this//;s/is//;s/a//;s/ *//g"**
Command c4	**perl -p -e 's/(^|\s+)is(\s+|$)//g;s/(^\|\s+)this(\s+\|$)//g;'**
Command c5	**perl -p -e 's/(^|\s+)(is|this)(\s+|$)//g;'**

功能	**converting string** **9600012301ADTA02ATDT03AATA04TADD05TDAA06ABCD** **into the following format:** **96000123** **01ADTA** **02ATDT** **03AATA** **04TADD** **05TDAA** **06ABCD**
Command c6	sed 's/\([0-9][0-9][A-Z][A-Z][A-Z][A-Z]\)/\ \1/g'
解釋	**It reads "insert newline before every occurrence of two numbers followed by four uppercase letters."** **If spaces/tabs on a line count as a blank line, then you need a more complicated expression.**

功能	**switch the first two columns**
sed/ex Script c7	**%s/^\([^]*\) \([^]*\)/\2 \1/**
解釋	注意 \(\) \1 \2 在 ex/sed 裡的意義
awk Script c8	**awk '{print $2 $1}'**
解釋	注意 ' ' 這對引號在這裡的用處，它可以避免**shell** 將**$1, $2** 當成變數。讀者應盡可能熟悉 ' ', " ", 及 ` ` 這幾對引號的用法。

功能	**convert a file into 2 coulumn**
Command c9	**pr -t -l66 -w80 -2 <filename>**
註	現在有 **Winword** 作爲文件的編輯程式，在**Unix**將文件做兩欄式的編排之需求已經大不如前。

5.4 檔案片段之擷取

功能	檔案片段之截取 (cut a block of lines)
用法	gl <beginline> <endline> <filename>
Script b1	sed -n -e "$1,$2p" $3
Script b2	QuitPoint=`expr $2 + 1` sed -n -e "$1,$2p ${QuitPoint}q" $3
解釋 b1,b2	使用 Script b1,b2 ，使用者要提供開始及結束之行數 Script b2 比 Script b1 快，當 awk 將輸入檔案讀入時，讀到 <endline> 時就停止，而Script b1 則會一直讀到最後一行，當檔案很大時會浪費時間。
Command b3	awk '/<beginline>/,/<endline>/' <filename>
解釋 b3	使用這個指令時使用者不是輸入行數，而是輸入含有 <beginline> 字串的行及含有 <endline> 字串的行。
Script b4	awk '{if ($0=="<beginline>") \ n=1 \ else { if($0=="<endline>") \ n=0 \ if (n==1) \ print}}' <filename>
解釋 b4	同上，但輸出時不包括 <beginline>以及<endline>這兩行。
註	請注意，這些 script 沒有檢查 <beginline> 及 <endline> 的大小，讀者自己要小心。

功能	delete a block of lines from a file
用法	dl <beginline> endline> <filename>
Script b5	cp $3 tmp.o echo "$1,$2d w" > script ex $3 < script
解釋	這個script 先製作一個編輯指令檔，再利用它來編輯原文，備份留在 tmp.o
註	請注意，這個 script 沒有檢查 <beginline> 及 <endline> 的大小，讀者自己要小心。

功能	delete all lines that contain some string and output to STDOUT
Command b6	grep -v "string" <filename>
註	請注意，這個 script 沒有檢查 <beginline> 及 <endline> 的大小，讀者自己要小心。

功能	delete blank lines from a file and output to STDOUT
Command b7	egrep -v '(^$)' <filename>
解釋	注意 -v 在 egrep 裡的作用，而 '^$' 在 regular expression 中代表空行

註	請注意，這個 **script** 沒有檢查 **<beginline>** 及 **<endline>** 的大小，讀者自己要小心。

功能	**replace a block of lines of a file**
用法	**rl <beginline> <endline> <filename> <filename>**
Script b8	`mv $3 tmp.o` `a=`expr $1 - 1`` `b=`expr $2 + 1`` `head -$a tmp.o > $3` `cat $4 >> $3` `sed -n -e "$b,\$p" tmp.o >> $3`
註	請注意，這個**script** 沒有檢查 **<beginline>** 及 **<endline>** 的大小，讀者自己要小心。

5.5 FTP 相關程式

功能	**batch ftp a file**

Script p1	`ftp $1 <<%`
	`lien`
	`binary`
	`put $2  (OR get $2)`
	`quit`
	`%`

功能	**batch ftp many files**

Script p2	`echo "ftp $1 <<%`
	`lien`
	`binary" > script`
	`for i`
	`do`
	`  echo "put $i"`
	`done >> script`
	`echo "quit`
	`%" >> script`
	`sh script`

5.6 簡單的統計運算

功能	**compute average of a column**
用法	**avg <column> <filename>**

Script ma1	`awk "`
	`  {sum += \$$1;}`
	`  END {print sum/NR}  "  $2`

功能	**compute the maximum of any column**
用法	**max <column> <filename>**

Script ma2	awk "
	BEGIN { MAX = -999999}
	{
	if (\$$1 > MAX)
	MAX = \$$1
	}
	END {print MAX}" $2

5.7 英文拼字工具

功能	**find an English word**
用法 （例）	**wordhelp con v en t** **(e.g. looking for the word "convenient")**
Script s1	pat=$1; shift grep "^$pat" /usr/lib/dict/all | mgrep $*
解釋	很多**Unix** 系統都有個**spell**的指令，這個指令一定會用到一個字典，其中含有大部分的英文單字，我們可以運用這個檔案來做很多跟字典有關的工作。使用者可以嘗試找找這個檔案 **/usr/lib/dict/all**。 **wordhelp** 這個**script** 是筆者最常用的**script**，寫英文文章時幫助非常大。我們對一個英文字的拼法不熟時，查一般的字典也無從下手，可能要試很多次，將一部字典翻來覆去才能找到要查的字。 **wordhelp** 這個 **script** 可讓使用者輸入一個英文字的某些片段，就可以將含有這些片段的英文字列出來，而同時含有這些片段的英文單字屈指可數，很容易就可挑出自己想找的單字。 請注意，這個**Script** 出現了一個罕見的 **shift** 指令，這是 **shell** 李一個重要的指令，可以將使用者在**command line** 下的參數做個變動，將 **$1** 捨棄，而將後面的 **$2, $3, $4** 等往前移動，如此，在**script** 中就可以將原代表所有參數的變數符號**"$*"**用來代表 **$2, $3, $4** 等參數。

功能	**grep with multiple patterns**
用法	**mgrep <pat1> <pat2> <pat3> ...**
Script s2	case $# in 0) cat ;; 1) cat | grep $1 ;; 2) cat | grep $1 | grep $2;; 3) cat | grep $1 | grep $2 | grep $3;; esac
解釋	**Script s2** 會將**STDIN**含有數個指定字串的「行」抓出來，使用者可輕易更改這個**script**，讓他可以**grep** 更多的**pattern**。對於常常使用的人，可省下許多時間。下面的 **spellcorrect** 會用到這個關鍵的 **script**。

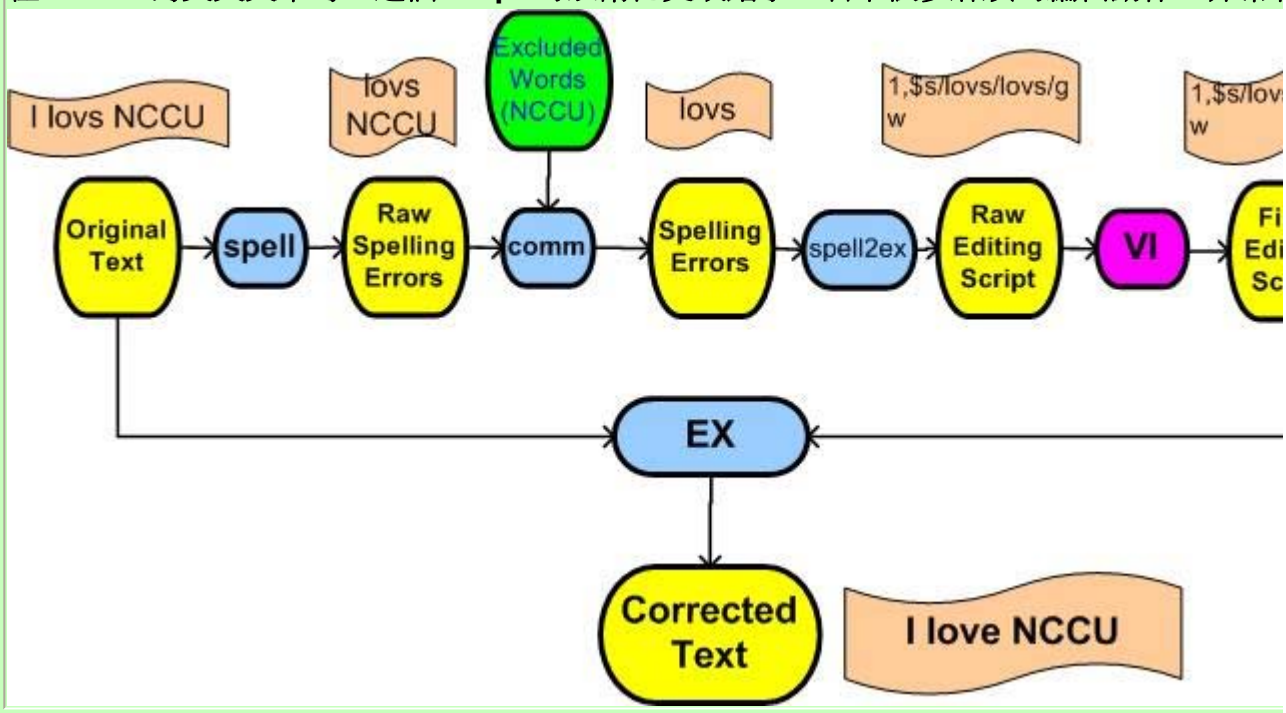
功能	**Spelling Corrector**
用法	**spellcorrect <filename>**

Script s3

```
spell $1 > 1.o
comm -23 1.o excludedwords > 2.o
sed 's/..*/1,$s\/&\/&\/g' 2.o > script
vi script
ex $1 < script
```

解釋

這個**script** 先將輸入的檔案利用**spell**找出字典沒有的字，其次將使用者的專有字剔除，然後
下的錯字製造出一個編輯指令檔，再叫出**vi** 來讓使用者將該檔案編輯成一個可以更正錯字的
指令檔，最後叫出**ex** 編輯器利用此編輯指令檔將原文裡的錯字改掉。當然，讀者可自行將**v**
爲自己喜歡的編輯器。
在**Unix** 上寫英文文章時，這個**script** 可以幫忙更改錯字，省下很多麻煩的編輯動作，非常有



問題

利用這個**script**時，有時候要注意編輯指令檔的正確性，萬一發生錯誤時，可能會破壞原檔
到不該改的字。例如，如果文章裡有個 **"th"**的字被挑出來，使用者如果將這個指令 **1,$s/th/**
成 **1,$s/th/the/g** 的話，後果將會很嚴重。

此外，**ex** 是一個 **atomic** 的程式， **(do all or do nothing)**，如果其中有一個編輯指令失敗
的 **w** 將不會作用，導致執行失敗。 例如，下面的編輯指令會失敗。

```
1,$s/systen/system/g
1,$s/systens/systems/g
w
```

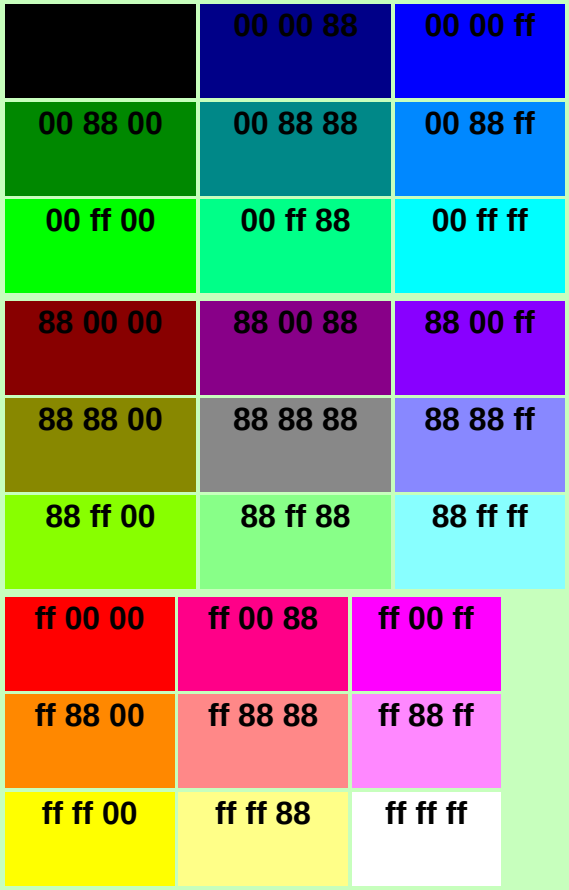
讀者應該在**editor** 內將編輯指令檔改成下面這樣才行：

```
1,$s/systen/system/g
w
```

5.8 HTML 相關程式

功能	**generate a color table**

```
for R in 00 88 ff
do
  echo "<TABLE>"
  for G in 00 88 ff
  do
    echo "<TR>"
    for B in 00 88 ff
```

Script h1	<pre> do echo "<TH bgcolor=#RG$B> <font size=3 color=BLACK>$R $G $B <font size=5 color=#$RGB>$R $G $B</TH>" done echo "</TR>" done echo "</TABLE>" done</pre>
解釋	本 script 會產生一段 HTML 碼，這段 HTML 碼在瀏覽器上會顯示一個色表，裡面有各種顏色及其對應的 RGB 碼，使用者可輕易改寫，增加更多的顏色。
結果	
註	這個 Script 裡 R,G,B 的參數如果改爲 00, 33, 66, 99, CC, FF，就可以產生一個展示216安全網頁顏色的 HTML 檔