大家來學 *Vim*

一個歷久彌新的編輯器

這份文件尚在草稿階段！！！

李果正 Edward G.J. Lee

Email: edt1023@speedymail.org

2003 **年** 3 **月** 3 **日**

目 錄

第 1 章

進來先看看

鑑於仍有許多人還沒找到順手的編輯器，而許多想學 vi 的人又覺得無從下手，因此在此提出一些個人的心得，希望能對這些朋友有點幫助。或許也減少一點 FAQ 吧！

真要深入的話，大多數的前輩都認為 vi 比 emacs/xemcas 還難學。但誰又真的需要熟悉編輯器的所有功能呢？你大可以邊用邊學啊！需要用到的先學，其它的就放一邊，只要能善用一些常用到的功能，又何必要那麼深入呢？而且您在使用當中經常會發現一些新功能，這又會馬上讓您給賺到了。

1.1 爲什麼選 *Vim* ？

最最重要的原因是可以正確處理中文！其它如 elvis, vile, nvi 在中文方面都會有問題(nvi-m17n 的版本已可以正確處理 Big5 中文，但功能仍不及 *Vim*[1] 完整)。另外就是許多作業系統都有 *Vim* 可用。當然如果您不需要中文支援的話，也建議使用 elvis[2]。vile 則有 emacs 的味道，而 nvi 大概是最忠於原味的了！至於原始 vi 的書已有中文翻譯（O'Reilly），各位到大一點的書局翻翻就有了。所以選定 *Vim* 做對象，兼述及 elvis，至少她不「排斥」中文。[3]

另一個原因，*Vim* 不僅是自由軟體（Free Software[4]），也是慈善軟體（CharityWare[5]），

[1] VIM 代表 Vi IMproved

[2] elvis 可直接讀 HTML(可用來上網，但只有文字)，binary、manpage 及 TEX/LATEX 檔(會自動去除一些指令！)，但不能真正「處理」中文就是了！

[3] *Vim* 也不是真的能完全支援中文啦！有些細部的功能還是沒法度。但 *Vim* 的發展群非常的活躍，一直都在改進中。平常一般的編輯動作應該是沒什麼問題。

[4] 什麼是自由軟體？請參考：http://www.gnu.org/philosophy/free-sw.html。

[5] 請參考在荷蘭的基金會網頁：http://iccf-holland.org。

如有贊助或評比得獎（幾乎是年年最佳編輯器獎 :），所得皆救助烏干達 AIDS 孤兒。您要使用當然是免費，您要捐款贊助當然是不勉強啦！但如果有評比有獎金可拿的，您去投她一票就是功德無量了。另外，如果想網路購書，可以透過 ICCF 的網頁 http://iccf-holland.org/click.html 來訂購，這樣 Amazon 會付一些贊助金給 ICCF，完全不會多花您一毛錢。

另外 *Vim* 的規則表示式(regular express)頗完整，您也可借這個機會學 regexp，因為您在 sed, awk, perl, less, grep... 中也是要用到，早點習慣 regexp，您生活在 Linux(Un*x)的世界會更美好。學了 regexp 您會有點看不起 Windows 系統中的找尋工具的。

由於是慈善軟體，廣結善緣，因此連中文繁體都支援，不簡單。但也因此最近的版本有點肥，但又不會太肥(比 xemacs 好多了啦！)。這麼可愛的軟體，能不用她嗎？

1.2 何處抓 *Vim*(elvis)？

http://www.vim.org/
http://vim.sf.net/download.php
ftp://ftp.vim.org/pub/vim/

找個自己中意的 mirro 站抓。或許也順便抓 Win32 的可執行檔回來在 windows 系統中使用。別忘了 runtime 檔也要抓，否則會無法找到需要的檔案來執行。

http://elvis.the-little-red-haired-girl.org/

也可試試 elvis（當然也是自由軟體）。

ftp://ftp.cs.berkeley.edu/ucb/4bsd/
ftp://ftp.foretune.co.jp/pub/tools/nvi-m17n/

nvi-m17n 也可以一試，目前已可以支援中文。

目前最新的版本，*Vim* 是 6.1，elvis 是 2.2h-beta，nvi 是 1.79。當然，本文的重點是放在 *Vim*。

1.3 編譯安裝

只講重點，避免囉嗦！

1.3.1 *Vim*

1. configure **時加上：**
   --enable-multibyte
   --enable-xim
   --enable-fontset
   --enable-gui=gtk
   --enable-perlinterp **需嵌入** perl **的話**
   --enable-pythoninterp **需嵌入** python **的話**
   GUI **可有好幾種選擇，但建議使用** gtk+**，比較穩定。**

2. ~/.vimrc **加入：**
   set enc=big5
   set guifontset=**英文字型，中文字型**
   設了 guifontset **就不能設** guifont **否則會優先使用**
   guifont**，這樣就找不到中文字型了！**console **下或** xterm
   下的話，就看您用什麼字型就顯示什麼字型，和 guifontset **無關。**

   英文字型要用固定字，我個人的實際設定例子(1024x768)**：**

   ```
   set guifontset=-b&h-*-medium-r-normal-*-*-120-*-*-m-*-*-*,
   \-arphic-*mingti2l*-medium-r-normal-*-*-190-*-*-c-*-big5-0
   ```

   Windows **中文版無需設** guifontset**，只要設** guifont **即可。例如：**

   ```
   set guifont=Andale_Mono:H11:W6
   ```

3. locale **要設成** zh_TW.Big5**，意思就是您的系統的** i18n
   支援要完整，我個人的設定如下：

   ```
   unset LC_ALL
   export LC_CTYPE=zh_TW.Big5
   ```

1.3.2 elvis

elvis **沒什麼好說的，**GUI **版本要靠靠** XA+CV **來顯示及輸入中文，因為他還不支援**

i18n/XIM。但在 console/rxvt 下則可以顯示、輸入中文。至於其他外觀調整，就請 man elvis。

1.3.3 nvi-m17n

nvi 是在 *BSD 系統的標準配備，m17n(multilingualization, 取前 m 及後 n 及其中的 17 個英文字母，合成 m17n) 則是日本朋友修改的版本，目前也可以用於中文，如果您系統上也有 nvi-m17n 的話，可以設定 /.nexrc 如下：

```
set noskipdisplay
set displayencoding=big5
set inputencoding=big5
set fileencoding=big5
set autodetect=tw
```

1.4 勤前教育

Vim 的基本模式（basic mode）可有六種：

1.4.1 常態模式（Normal mode）

您一進入 vim 就是處於這常態模式，只能下按鍵指令，不能輸入編輯文字。這些指令可能是游標移動的指令，也可能是編輯指令或尋找替換指令。

1.4.2 插入模式（Insert mode）

按 i 就會進入插入模式，此時才可以鍵入文字，寫您的文章，按 Esc 又會回到正常模式。此時在狀態列會有 -- INSERT -- 字樣。

1.4.3 命令列模式（Cmdline mode or Command-line mode）

按冒號：(別忘了 Shift 鍵)就會進入命令列模式，左下角會有一個冒號：出現可下 ex[6]指

[6] vi, ex, sed 都是衍化自 ed。ed 是一個很古老的行編輯器，就好像 DOS 下的 edline 一樣。當然 DOS

令。也是按 Esc 回命令列模式。反正正常狀態都是處於常態模式，這樣才不會把您辛苦打字的文章給隨便搞亂掉。搜尋時的 / 及 ? 按鍵也是屬於命令列模式。

1.4.4 反白模式（Visual mode）

這就是把需要處理的文字反白起來，等待處理的模式，底部狀態最會有 `-- VISUAL --` 或 `-- VISUAL LINE --` 或 `-- VISUAL BLOCK --` 字樣。詳細會在第 7 章，頁 27，做介紹。

1.4.5 選擇模式（Select mode）

和反白模式的異同待瞭解。

1.4.6 Ex 模式（Ex mode）

這和命令列模式是一樣的，但下了命令後不會返回，會留在行編輯器 ex 的狀態，有一個冒號在那兒，等待輸入命令。由 gQ 按鍵可以進入 Ex 模式，輸入 visual 可以返回正常模式。

其它還有五種的額外模式（additional mode）！這裡不準備說明，會在第 3.1 節，頁 11 做介紹。反正，使用 *Vim* 的無上心法就是，有問題先考慮按 Esc 鍵回到常態模式再說。

1.5 基本教練：step by step

1.5.1 由命令列來開檔

`vim test.txt` 或 `gvim test,txt`，如果您的系統 `vi` 是 `vim` 的連結檔話，就可以直接用 `vi test.txt`。以下就直接用 `vim` 代表 `vi`，`gvim`，`elvis`，因操作是一樣的有不同的地方會加注說明。

下的 edline 也是學 ed 而來的，但功能可就不能同日而語了！有興趣可 man ed 學看看，有些時候會只有 ed 可以用（當機救助的時候）。ex 的操作和 ed，除了多了個冒號外，幾乎是一樣的。

1.5.2 先進入 vim 後再開檔

進入 vim 後，使用冒號命令 :e test.txt，就可以編輯 test.txt 這個檔。第 1.5.1 小節及第 1.5.2 小節這兩個開檔法，如果 test.txt 不存在的話，就會開一個以 test.txt 為名的新檔案。

如果是 gvim，可由 icon(gtk 版本才有)或功能表來叫出 file browser 來選看看您要編輯哪一個檔，但如果您是初學的話不建議您這麼做，vi 就是以按鍵快速聞名，這是她的優點，您還是學起來吧，不然沒有 GUI 的時候會很不習慣的。

1.5.3 編寫文件

進入 vim 後，按 i 進入插入模式，就可以編寫您的文件了。在 vim 游標的移動可以由方向鍵來移動。Backspace 鍵可消去前一個字元，中文的話是一個中文字。Del 鍵可刪除游標所在處的字元（中文字）。

原始 vi 是不能在插入模式隨意移動游標的，得進入正常模式才能移動，因此就常常要按 Esc 來變換模式。vim 及 elvis 都打破了這個規矩。

1.5.4 存檔、離開

如果您寫好您的文件，就可以按 Esc 回到正常模式，然後 :w 就會存檔（注意，是冒號命令），但還不會離開 vim，要離開可按 :q，就可以了！也可以合起來用，:wq，就樣就會存檔後離開。怎麼樣，也不會很難吧！只不過操作方式和別的編輯器不一樣罷了，這樣豈不是很有個性。:-)

盡量記住按鍵的意義，才不必死背，如 e 是 edit(編輯)，w 是 write(寫入)，q 是 quit(停止、離開)。

這裡要提醒大家一下，許多 distributions 中會編譯一個小型的 vim，啟動會比較快一點，但缺乏許多本文要用到的功能，因此，建議您使用 vim/gvim，而暫時避免使用 vi，或者就把 vi 直接連結到正常的 vim 上去。Slackware Linux **的話，他的 vi 是連結到 elvis 的，也請使用 vim/gvim 為指令，或改變 vi 的連結。而 *BSD 系統，使用的很可能就是 nvi，這些請使用時注意一下，以免和文中內容所述不符。**

如果不確定自己是使用哪一種版本的 vi，可以進入 vi 後按冒號 : 後再輸入 ver，然後按 Enter，就會得知是哪一種版本的 vi，如果是 vim 的話，還會顯示前有 +/–號的各功能，

有 + 號的，表示有編譯進去，–號的表示沒有這項功能。

好了，這是就編輯的整個過程。下回開始是詳述各部份的功能，把 *Vim* 解剖開來講，您可以馬上現學現賣。由於 `vi/vim` 的操作方式很有個性，因此，用了一次就會記住有這麼一個功能，想忘也忘不了，但有時按鍵難免會忘記，有這麼一種功能大概是忘不了的，查了幾次指令就可以記得住了。

第 2 章

游標的移動

本節所述皆是在 common-mode(c-mode，**在** *Vim* **又名** normal-mode，**就是剛進入** vim，**不能輸入文字的狀態**)**下的移動，原始的** vi **只能在** c-mode **移動游標，在** insert-mode **只做文字的輸入，而不做游標的移動。當然** *Vim* **及** elvis **的方向鍵是不論在那一種** mode **皆可移動自如。**

2.1 基本的游標移動

h	**左，或** Backspace **或方向鍵。**
j	**下，或** Enter **或** +**（要** Shift **鍵），或方向鍵。**
k	**上，或 方向鍵或** –**（不必** Shift **鍵）。**
l	**右，或** Space **或方向鍵。**
Ctrl+f	**即** PageDown **翻頁（**Forward**，向前、下翻頁）。**
Crtl+b	**即** PageUp **翻頁（**Backward**，向後、上翻頁**)**。**

使用 hjkl **鍵的移動是為了使手不必離開打字區（鍵盤中央的部位），以加快打字的速度，如果各位不習慣，那就使用方向鍵吧！其實，一旦習慣了以後，對於編輯工作的效率會有很大的幫助，而且有許多工作站的** vi **只能使用** hjkl **的移動方式，因此可能的話，盡量熟悉** hjkl **的游標移動。**

Backspace **及** Space **的移動方式是到了行首或行尾時會折行，但方向鍵或** hl **鍵的移動則在行首或行尾時您繼續按也不會折行。轉折換行的功能是** *Vim* **的擴充功能，**elvis **無此功能。**

jk **及使用方向鍵的上下移動游標會盡量保持在同一欄位。使用** Enter，+，–**的上下移**

動，游標會移至上（下）一行的第一個非空白字元處。

好像有點複雜，各位就暫時使用方向鍵來移動就簡單明白了！等您愛上了 *Vim* 後再來講究吧。

2.2 進階的游標移動

0 是數目字 0 而不是英文字母 o。或是 Hmoe 鍵，移至行首，（含空白字元）。
^ 移至行首第一個非空白字元，注意，要 Shift 鍵。
$ 移至行尾，或 End 鍵。要 Shift 鍵。
G 移至檔尾（全文最後一行的第一個非空白字元處）
gg 移至檔首（全文第一行之第一個非空白字元處）。

在規則表示式（regular expression）中，^ 是匹配行首，$ 是匹配行尾。

gg 是 *Vim* 的擴充功能，在 elvis 或原始 vi 中可用 1G 來移至檔首（是數字 1 不是英文字 l）。G 之原意是 goto，指移至指定數目行之行首，如不指定數目，則預設是最後一行。

w 移至次一個字（word）字首。當然是指英文單字。
W 同上，但會忽略一些標點符號。
e 移至後一個字字尾。
E 同上，但會忽略一些標點符號。
b 移至前一個字字首。
B 同上，但會忽略一些標點符號。
H 移至螢幕頂第一個非空白字元。
M 移至螢幕中間第一個非空白字元。
L 移至螢幕底第一個非空白字元。這和 PageDown，PageUp 不一樣，內文內容並未動，只是游標在動而已。
n| 移至第 n 個字元(欄)處。注意，要用 Shift 鍵。n 是從頭起算的。
:n 移至第 n 行行首。或 nG。

2.3 特殊的移動

) **移至下一個句子（**sentence**）首。**

(**移至上一個句子（**sentence**）首。**sentence**（句子）是以** . ! ? **為區格。**

} **移至下一個段落（**paragraph**）首。**

{ **移至上一個段落（**paragraph**）首。**paragraph**（段落）是以空白行為區格。**

% **這是匹配** {}**，**[]**，**() **用的，例如游標在** { **上只要按** %**，就會跑到相匹配的** } **上。**

另還有一些 *Vim* **的特殊按鍵，但這得留待最後再來說明，否則各位恐怕會頭昏眼花了。**

第 3 章

基本編輯指令

這個單元就開始進入主題了。下編輯指令都是在常態模式，就是您一進入 *Vim* 時的模式，只能下指令，不能鍵入文字。如果印象模糊，請瞄一下第一個單元的內容。這個單元說的是基本的編輯指令，有些比較特殊的編輯指令，因為太有個性了，所以會獨立成一個單元來說明。

3.1 五種額外模式（additional mode）

這裡要對基本模式（請參考第 1.4 節，頁 4）外的五種額外模式做介紹，以下的章節會有機會碰上。為什麼會有這麼多模式？這樣豈不是很易就搞混？其實一般編輯不必太在意一些模式，您只要在各種模式下親自去操作幾次就會知道這些模式的作用，不必特意去記憶他，這些模式和基本模式都有連帶關係的。

3.1.1 操作等待模式（operator-pending mode）

這其實和一般的常態模式一樣，只不過是指在常態模式下了某些編輯指令，等待其他動作的狀態。

3.1.2 取代模式（replace mode）

指下 R 指令時所處的狀態。請參考第 3.4 節，頁 13。在狀態列會有 `-- REPLACE --` 字樣。

3.1.3 插入常態模式（insert normal mode）

這是一個很特殊的模式，在插入模式時，進入輸入狀態，但按 Ctrl+O **就會進入插入常態模式，和常態模式一樣，只不過執行完所下的指令後又會馬上返回原來的插入模式繼續輸入文字。狀態列會有** `-- (insert) --` **字樣，是小寫有小號的。**

3.1.4 插入反白模式（insert visual mode）

這和插入常態模式一樣，只不過在按 Ctrl+O **後所執行的是反白的** Ctrl+V **或** V **或** v **而進入反白模式，等反白模式結束又會返回原來的插入模式。狀態列會有** `-- (insert) VISUAL --` **字樣。**

3.1.5 插入選擇模式（insert select mode）

這和插入反白模式一樣，只不過進入的是選擇模式，而非反白模式。狀態列會有 `-- (insert) SELECT --` **字樣。**

3.2 進入插入模式的指令

- **i** **在游標所在字元前開始輸入文字**(insert)**。**
- **a** **在游標所在字元後開始輸入文字**(append)**。**
- **o** **在游標所在行下開一新行來輸入文字**(open)**。**
- **I** **在行首開始輸入文字。此之行首指第一個非空白字元處，要從真正的第一個字元處開始輸人文字，可使用** 0i **或** gI(*Vim* **才有**)**。**
- **A** **在行尾開始輸入文字。這個好用，您不必管游標在此行的什麼地方，只要按** A **就會在行尾等著您輸入文字。**
- **O** **在游標所在行上開一新行來輸入文字。**
- **J** **將下一行整行接至本行**(Joint)**。**

並無相對的 split **功能，可在插入模式下按** Enter **來達成，當然如果您熟** macro **的話，可自行定義。使用** J **時，預設會消去本行的** EOL(End Of Line) **字元，且上下行接縫間會留下一個空白字元，這符合英文習慣，卻對中文會造成困擾，欲不留空白字元，可使用** gJ**（大寫** J**） 指令，但這是** *Vim* **的擴充功能，**elvis **不適用。要和中文相容，可參考底下會說明的重排功能的** *Vim* script**。 請您隨便找一個檔案來試看看，光看文字說明太抽象了。**

3.3 刪除指令

x 刪除游標所在處之字元，在中文指一個中文字。在 *Vim* 及 elvis 亦可用 Del 鍵。

X 刪除游標前之字元。不可使用 Backspace 鍵，除非是在插入模式。*Vim* 可以正確使用以上兩個指令於中文，會刪去一個中文字。elvis 則不行，一個中文字要刪兩次，即使用 xx。

dd 刪除一整行(delete line)。

dw 刪除一個字(delete word)。不能適用於中文。

dG 刪至檔尾。

d1G 刪至檔首。或 dgg(只能用於 *Vim*)。

D 刪至行尾，或 d$（含游標所在處字元）。

d0 刪至行首，或 dˆ（不含游標所在處字元）。請回憶一下 $ 及 ˆ 所代表的意義，您就可以理解 d$ 及 dˆ 的動作，這就是 `vi(m)` 可愛之處。

3.4 取代及還原

r 取代游標所在處之字元。

R 進入取代模式（replace mode），取代字元至按 Esc 為止。

cc 取代整行內容。或大寫 S 亦可。

cw 替換一個英文字(word)，中文不適用。(change)

~ 游標所在處字元之大小寫互換。當然不能用於中文。別忘了 *Shift*！

C 取代至行尾，即游標所在處以後的字都會被替換。或 c$。

c0 取代至行首，或 cˆ。

s 替換一個字元為您所輸入的字串。和 R 不同，R 是覆蓋式的取代，s 則是插入式的取代，您可親自實驗看看。ㄟ！是小寫的 s。

u 這個太重要了，就是 undo，傳統的 vi 僅支援一次 undo，*Vim* 及 elvis 就不只了，*Vim* 幾乎是沒有限制的。

U 在游標沒離開本行之前，回復所有編輯動作。

Crtl+r 這個也是很重要，就是 redo 鍵。

Vim 很有個性的，您在常態模式按了 r 她就會停在那裡等主人鍵入所要替代的字元，希望您這個當主人的，不要傻呼呼的也楞在那裡，趕快鍵入您的新字元吧！:-) *Vim* 中可用於中文字，也就是可以替換一個中文字，elvis 則不行。當然您的 *Vim* 是要設在 big5/cp950

的才行。怎麼樣！有沒有看過如此有個性的取代方式？丫！r 就是 replace 啦！

3.5 加上數目字

喔！騷到 *Vim* 的癢處了，這是 *Vim* 一個非常騷包的功能，只此一家別無分號（當然同源的 ed，sed 等不在此限）。就是您可以在大部份的指令前加上數目字，代表要處理幾次的意思。以下用實例來說明比較清楚。

5dd	刪除游標所在處（含）起算以下五行內容。妙吧！
3r	按了 3r 後，您鍵入一個英文字，則三個字元皆會被您所鍵入的英文取代。只要 locale 設定正確，中文也通喔！
5J	將五行合併成一行。
3x	刪除三個字元。中文也通。
5i A	然後按 Ecs，插入五個 A。中文也可以！
2i sys Esc	插入 syssys。中文也可以！
5G	游標移至第五行，是從檔首開始起算。和 :5 作用相同。
5l	移至右第五個字元處，當然 j 是可以用方向鍵取代的。

所有移動指令（參考第 2 章，頁 8）都可以加上數目字來控制，中文也通喔！其它的指令和數目字結合，就留待各位去發掘吧！最重要的是請您親自操作看看，使用 *Vim* 常常要動動腦筋，會有更妙的操作方式，想一次可以用很久喔！有人說，學電腦的人，動腦筋就是為了偷懶。:-)

3.6 簡單重排功能

>>	整行向右移一個 shiftwidth（預設是 8 個字元，可重設）。
<<	整行向左移一個 shiftwidth（預設是 8 個字元，可重設）。:set shiftwidth? 可得知目前的設定值。:set shiftwidth=4 可馬上重設為 4 個字元。shiftwidth 可簡寫成 sw。ㄟ，別忘了 Shift 鍵！
:ce(nter)	**本行文字置中。注意是冒號命令！**
:ri(ght)	**本行文字靠右。**
:le(ft)	**本行文字靠左。所謂置中、靠左右，是參考** textwidth(tw) **的設定。如果** tw **沒有設定，預設是**80**，就是以** 80 **個字元為總寬度為標準來置放。當然您也可以如** sw **一樣馬上重設。**
gqap	**整段重排，或** gqip**，在段落中位何地方都可以使用。和中文的配合見下述。**
gqq	**本行重排。**
gqQ	**全文重排，是以游標所在處的段落開始重排至檔尾。以空白行為段落的間隔。**

重排的依據也是 textwidth**。這裡的重排是指您鍵入文字時沒有按** Enter **鍵，就一直在**keyin**，這樣會形成一個很長的一行（雖然螢幕上會替您做假性折行），重排後，則會在每一行最後加入** EOL**。**gq **重排功能是** *Vim* **才有的功能。**

如果是利用 visual mode **所標記起來的部份，只要按** gq **就會只重排被標記的部份。請參考第七章的標示指令一節。**

基本上 gq **就是一個獨立的重排指令，就像** d **或** y **是獨立的刪除、複製的指令一樣，所以，當然是可以加上數目字加以控制，或和其他指一起用的，例如：**

gq3q	➪	**重排三行**
gq2ap	➪	**重排兩個段落**
gq5j	➪	**重排游標以下五行(別忘了** j **是向下移動，**5j **就是向下移動五行，包括游標所在處就是六行**)
gq}	➪	well**，這是什麼哇？**
gq)	➪	**這又是啥麼哇？請複習一下第** 2 **章的第** 2.3 **特殊移動一節。並親自做一下實驗看看。在中文文稿，通常就是一個段落。**

重排的功能本不是編輯器的主要功能，而是文書排版軟體的工作，但簡單的重排也是很方便，對於中文而言，處理上得多花些工夫，對於希望有中文重排功能的朋友，可下載

Vim script 來使用：

http://info.sayya.org/~edt1023/vim/format.vim

這是修改自日本朋友的貢獻。把他置於 $VIMRUNTIME/plugin 目錄下，重新開啟 vim 就會生效，對於 J 及 gqap 都會考慮中文的細節。當然，*Vim* 只是個文字編輯器，如果要做進一步的排版，需要由 office 類的文書處理軟體，或更進一步的專業 TEX/LATEX/texinfo/troff/groff 排版軟體來處理。

第 4 章

複製（yank）

yank 是什麼意思？有疑問的請查一下字典吧！就好像是中醫治療中的「拔罐」的意思啦（是不是叫「拔罐」？知道的朋友指正一下吧）！反正在 *Vim* 中，她就是複製 copy 的意思。這在 *Vim* 的思考邏輯裡，就是「拔」yank 起來，「放」put 上去。其實複製的指令就是 y 一個而已，為什麼要獨立成一個單元來說明呢？因為 *Vim* 複製、貼上的功能實在太獨特了，再配合第三單元介紹的數目字，及 *Vim* 內部的緩衝區來使用的話，您會發現，原來 *Vim* 肚子裡還暗藏著秘密武器。

4.1 指令說明

yy 複製游標所在行整行。或大寫一個 Y。

2yy 複製兩行，y2y 也可以。ㄟ，請舉一反三好不好！:-)

y^ 複製至行首，或 y0。不含游標所在處字元。

y$ 複製至行尾。含游標所在處字元。

yw 複製一個 word。

y2w 複製兩個字。

yG 複製至檔尾。

y1G 複製至檔首。

p 小寫 p 代表貼至游標後（下）。

P 大寫 P 代表貼至游標前（上）。整行的複製，按 p 或 P 時是插入式的貼在下（上）一行。非整行的複製則是貼在游標所在處之後（前）。

"ayy 將本行文字複製到 a 緩衝區。

a 可為 26 個英文字母中的一個，如果是小寫的話，原先的內容會被清掉，如果是大寫

的話是 append 的作用，會把內容附加到原先內容之後。" 是 Enter 鍵隔壁的那一個同上符號（ditto marks），當然是要和 shift 鍵同時按的。

"ap 將 a 緩衝區的內容貼上。

這個緩衝區的術語在 *Vim* 稱為 registers，*Vim* 擴充了相當多的功能。您用 d、c、s、x、y 等指令改變或刪除的內容都是放在 registers 中的。例如：您用 dd 刪除的一行，也是可以使用 p 來貼上的。只要是在緩衝區的內容都可以使用 p 來貼上，不是一定要 y 起來的內容才能用 p。因此您認為 p 是 paste 也可以，認為是 put 可能較正確。

5"ayy 複製五行內容至 a 緩衝區。
5"Ayy 再複製五行附在 a 內容之後，現在 a 中有十行內容了！

ㄟ！不要我一直用 a 您就認為只有 a 可以用喔。26 個英文字母都可以的，交叉運用下，您會發覺 *Vim* 肚量不小。

問題來了！忘記誰是誰的時候怎麼辦？ :reg（冒號命令）就會列出所有 registers 的代號及內容。您現在就試著按看看。咦！怎麼還有數目字、特殊符號的緩衝區，原來您剛剛刪除（複製）的內容就預設放在 " 這個緩衝區，然後依序是 0,1,2,...9。也就是說您按 p 不加什麼的話，是取出 " 緩衝區的內容。% 指的是目前編輯的檔案，# 指的是前一次編輯的檔案。還有其它的呀！這會在下一節做介紹。

Tab 補全的功能，elvis 也有，但叫出 registers 列表的命令則沒有，您得自行記憶在您的腦袋瓜子裡。而且 elvis 的補全能力並沒 *Vim* 強。

4.2 Register 緩衝區

在 *Vim* 裡頭，有許多不同種類的緩衝區，例如：置放一整個檔案的 buffers 緩衝區（請參考第 6.2 節，頁 23）；檔案內容操作，如刪除、yank、置換，給 **P**ut 要用的 registers 緩衝區；另外還有給書籤要用的 marks 緩衝區（請參考第 5.4 節，頁 21）。雖然這些內容不一定是放在 RAM 記憶體內，有的是置於硬碟檔案上，需要時才從檔案存取，但這裡通通把他當做是緩衝區，以方便理解。

4.2.1 register 的種類

4.3 天大的指令

. 這是什麼？丫，是英文句點啦！沒錯，就是英文句點。什麼意思？重複前次的編輯動作。這個指令太高明了，只要是編輯動作（移動游標不算，冒號命令也不算）都可以按英文句點來重複，要重複幾次都可以。

例如：您按了 yy**，然後按** p **就會複製、貼上一整行，如果要重複這個動作的話，就可以按** . **，也可以把游標移到其它地方後再按。其它** dd**，**dw**，**r**，**cw **等編輯指令都可以這樣來重複。如果您要重複做某些編輯動作時，千萬千萬一定要想到有這麼一個英文句點重複指令。丫，拜託啦！您一定要常用這個指令。**

4.4 疑難雜症

4.4.1 那 mouse 中鍵的剪貼功能還有嗎？

當然還有，不管在 console **或** X terminal **中都照用不誤。當然在** windows **下的話就不能用了，可以用** Shift+Insert **來代替。**Ctrl+v **在** *Vim* **中另有作用，在** windows **下就不必去麻煩它了。**

4.4.2 軟體間互相 copy 時，常常都搞得天下大亂耶！

要設成 :set paste**。這是** *Vim* **的擴充功能，**elvis **沒有。那在** elvis **怎麼辦？只好** :set noai[1] **了。在** GUI **的版本應不會有這種情形。**

朋友！您睡著了嗎？不要被嚇到了，您只要開個檔案，親自操作一下就能心領神會。那用 mouse **不是更方便嗎？不見得，**yyp **來複製貼上一整行比較快，還是用** mouse **來拉比較快？您可以試看看。**

[1] set **的功能先不必去理它，會在第九章專門討論。**

第 5 章

搜尋、替換

搜尋、替換的功能幾乎是每個編輯器必備的功能，那在 *Vim* 中有沒有特殊的地方呢？當然有，您忘了，*Vim* 是個性十足的編輯器。最特殊的地方是和規則表示式（regular expression, **簡稱** regexp**）結合在一起。簡單的說她是一種** pattern **的表示法，在執行動作，如搜尋或替換時，就會依據這個** pattern **去找，所有符合** pattern **的地方就會執行您所下的動作。在這個單元裡暫不討論** regexp**，會在第十章來探討，以免搞得頭昏腦脹。目前就暫不使用** regexp**，您要找什麼就直接鍵入什麼就對了。**

5.1 搜尋

/ **在** c-mode **的情形下，按 / 就會在左下方出現一個 /，然後鍵入您要尋找的字串，按個** Enter **就會開始找。**? **和 / 相同，只是 / 是向前（下）找，** ? **則是向後（上）找。**

n **繼續尋找。**

N **繼續尋找（反向）。**

5.2 更方便的搜尋操作（*Vim* 才有）

***** **尋找游標所在處之** word**（要完全符合）。**

**同上，但 * 是向前（下）找，# 則是向後（上）找。**

g* **同 * ，但部份符合即可。**

g# **同 # ，但部份符合即可。**n, N **之繼續尋找鍵仍適用。**

5.3 替換（substitute）

```
:[range]s/pattern/string/[c,e,g,i]
```
[1]

range	**指的是範圍，**1,7 **指從第一行至第七行，**1,$ **指從第一行至最後一行，也就是整篇文章，也可以** % **代表。還記得嗎？** % **是目前編輯的文章，**# **是前一次編輯的文章。**
pattern	**就是要被替換掉的字串，可以用** regexp **來表示。**
string	**將** pattern **由** string **所取代。**
c	confirm**，每次替換前會詢問。**
e	**不顯示** error**。**
g	globe**，不詢問，整行替換。**
i	ignore **不分大小寫。**

g **大概都是要加的，否則只會替換每一行的第一個符合字串。可以合起來用，如** cgi**，表示不分大小寫，整行替換，替換前要詢問是否替換。**

[**實例**] `:%s/Edwin/Edward/g`
這樣整篇文章的 Edwin **就會替換成** Edward**。**

更進階的搜尋、替換的例子在說明 regexp **的時候還會再詳述。目前只知道最基本的用法就可以了！其實光這樣就非常好用了。**:-)

5.4 書籤功能

這又是 *Vim* **的一個秘密武器，簡單的說，您可以在文章中的某處做個記號（**marks**），然後跑到其它地方去編輯，在呼叫這個** mark **時又會回到原處。妙吧！**

mx	x **代表** 26 **個小寫英文字母，這樣游標所在處就會被** mark**。**
‘x	**回到書籤原設定位置。‘ 是** backward quote**，就是** Tab **鍵上面那一個。**
’x	**回到書籤設定行行首。’ 是** forward quote**，是** Enter **鍵隔壁那一個。**

這裡舉個簡單的子，請隨便開一個現成的檔案，把游標移到任一個位置，然後按 ma **做個** mark**，再按大寫** G **移到檔尾，然後按** ’a **看現在在什麼地方？**

[1] **方括號代表這個設定項是可以省略的，往後的章節同此解釋。如果寫成** `:opt[ion]`**，表示鍵入** `:opt` **及** `:option` **兩者皆可。**

5.5 *Vim* 對於書籤的擴充功能

5.5.1 小寫英文字母

只作用於單一檔案內。

5.5.2 大寫英文字母

可作用於各檔案間。例如 mA **會在** viminfo **中紀錄下這個檔案及位置，結束** vim**，然後再啟動** vim**，按** 'A **就會回到當初做標記的那個檔案及所在位置(**vim **會自動開啟做有** A **標記的檔案)。別懷疑，請自行馬上做個實驗就知道啦！**:-)

5.5.3 阿拉伯數目字

可作用於前次編輯的十個檔案。數目字的用法比較特殊，'0 **是回到前一次編輯檔案中離開前的最後位置，**'1 **則是回到前二次編輯檔案的最後位置，依此類推。您不必使用** m **來標示，**vim **會自動記憶。很玄吧！其實這是** viminfo **的功能，您要認真追究的話，請** :h viminfo-file-marks**。**viminfo **關掉，就沒這個功能了！所謂前次指的是前次啟動的** vim**。不管是哪一種的書籤，到達** mark **處(或檔案)，想返回原來的位置(或檔案)，可以按** Ctrl + O**。**

5.5.4 :marks

得知目前所有書籤的列表。

第 6 章

叫檔、存檔、緊急回復

ㄟ，是不是在灌水呀！怎麼開個檔也成一個單元？那您就錯了，在 *Vim* **裡叫檔的花樣可多了，而且又可以多檔編輯，各編輯中的檔案還可以互通訊息，這裡面學問可大著呢！** *Vim* **就更騷包了，也學人家檔案可以加密，雖說是噱頭，但也還滿好用的。**

6.1 開檔的一些花招

vim + **檔名**	**這樣開檔後，游標會落在檔案最後一行的行尾，在檔案屁屁後幹什麼呢？方便您可以繼續編輯嘛！**:-)
vim +n **檔名**	**開檔後，游標會落在第** n **行的行首。**
vim +/string **檔名**	**還記得嗎？** / **就是尋找指令，這樣進入檔案後游標就會落在第一個找到的** string **上，還可以按** n **繼續找** string **哦！喔，**string **還可以使用** regexp **來表示喔。**

6.2 多檔編輯

多檔編輯會有兩種情形，一種是在進入 vim **前所用的參數就是多個檔（這種情形稱為** argument list**）。另一種情形是進入** vim **後另外再開其它的檔（稱為** buffer list**）。不過都可以統稱為** buffer**。**

:n	**編輯下一個檔案。**
:2n	**編輯下二個檔案。**
:N	**編輯前一個檔案。注意，這種用法只能用於** argument list **的情形。**
:e *檔名*	**這是在進入** vim **後，在不離開** vim **的情形下再開其它檔案。只要您要編輯的檔案是在目前目錄，**Tab **補全鍵還是可以使用。**
:e# *或* **Ctrl+^**	**編輯前一個檔案，用於兩檔互相編輯時相當好用。這種用法不管是** argument list **或** buffer list **檔案間皆可使用。還記得嗎？#** **代表的是前一次編輯的檔案。**
:files	**或** :buffers **或** :ls**，會列出目前** buffer **中的所有檔案。在** elvis **中可使用** :b **來叫出** buffers**。在** buffers **中，減號** –**表示這個** buffer **並未載入，不過，不必擔心，載入相當快速的。加號** + **表示這個** buffer **已經修改過了。#** **代表前一次編輯的檔案，**% **是目前編輯中的檔案，這兩個代號應該很熟悉了吧！**
:bn	buffer next**。編輯次一個** buffer **的檔案。**
:bp	buffer previous**。編輯前一個** buffer **的檔案。**
:bl	buffer last**。編輯** buffer **中最後一個檔案。以上兩個指令** elvis **不適用。**
:b*檔名或編號*	**移至該檔。**

在 :ls **中就會出示各檔案的編號，這個編號在未離開** vim **前是不會變的。這個指令在** elvis **也是可以使用。當然** :e#**編號 也是可以的，這樣的用法則是所有** vi clone **都通用了。如果您是使用** *Vim* **的** GUI**，那就在功能表上就會有** Buffers **這個選項，可以很容易的知道及移動各** buffer **間。**

:bd(elete)	buffer **在未離開** vim **前是不會移除的，可使用這個指令移除。其實移除她幹什麼呢？**vim **是您在叫用時才會載入的，因此這些** buffers **並不是像** cache **一般要佔記憶體的。**
:e! *檔名*	**這樣也是會開檔，但會放棄目前編輯檔案的改變，否則如果檔案已有變動，**vim **預設是不讓您隨便離開的。**:e! **後不接什麼的話，代表捨棄一切修改，重新載入編輯中檔案。**
:f *或* **Ctrl+g**	**顯示目前編輯的檔名、是否經過修改及目前游標所在之位置。**
:f *檔名*	**改變編輯中的檔名。**(file)
:r *檔名*	**在游標所在處插入一個檔案內容。**(read)
:35 r *檔名*	**將檔案插入至** 35 **行之後。**
gf	**這是** vim **的特殊叫檔法，會叫出游標所在處的** word **為名的檔案，當然，這個檔案要在目前目錄內，否則會開新檔案。**

哦！好像有點給他複雜，主要原因是偶文筆不好啦！不過您何不選個順手的來用就可以了，選定了，以後就是使用他，這樣就不會那麼複雜了。:-)

6.3 離開

:q	**如本文有修改而沒存檔，會警告，且無法離開。**(quit)
:q!	**捨棄所有修改，強迫離開。**
:wq	**存檔後離開。縱使檔案未曾修改也是會再存一次檔。**
:x	**也是存檔後離開，但如果檔案沒有修改，則不會做存檔的動作。**
ZZ	**和** :x **完全一樣，隨您高興用哪一個。**
:w *檔名*	**另存他檔。不加檔名就是寫入原檔。**(write)

:q **及** :q! **是對目前編輯中的檔案作用，如果多檔編輯的情形並不會離開** vim**，這時可下** :qa **或** :qa! **來整個離開** vim**。**a **就是** all **的意思。**:**指令**!**，這個** ! **的意思是強迫中止目前正在編輯的動作，而去執行所下的指令。各位應該到目前為止碰過好幾次了吧！**

6.4 *Vim* 的加密功能

6.4.1 vim -x [檔名]

這樣進入 vim **後會要求輸入密碼。以後加密過的檔案由** vim **開啟時會自動要求輸入密碼。否則無法開啟。其它的編輯器當然是無法開啟的。**

6.4.2 進入 vim 編輯檔案中，可用 :X 指令

小心！vim **一開檔就會有個 .檔名**.swp **這個檔，是為了緊急回復用的，一般是在您所開檔案的所在目錄，這是個隱藏檔，**ls **要有** -a **參數才看得到，您加密的功能並沒有作用在這個** swp **檔，因此** root **還是知道您在寫些什麼關於他的壞話的。**:-)**當然啦！山不轉，路轉，路不轉，人轉，您也是可以把** swap **的功能關掉的** :set noswf **就行了！但如果您編輯的是大檔案，則不建議您把** swap **關掉，這樣會很吃記憶體的。**

elvis **的話，其暫存檔是統一集中存放在** /var/tmp/*.ses**，權限是檔案所有者始能讀寫。** *Vim* **的早期版本，其** *.swp **檔是依原檔案的權限來設定的，最近的版本則從善如流，已經改成檔案所有人始能讀寫，就是** -rw------- **啦！**

6.5 緊急回復

`vim -r` **檔名，或進入** `vim` **後，下** `:recover` **檔名，來回復。**

第 7 章

各種標示方法及視窗操作

這一個章多了一種模式，那便是 visual mode**（以下簡稱** v-mode**）**v-mode **下的反白區（反黑區？^_^）在本文就統一稱為標示區，不知各位是否有更好的中文名稱？ㄟㄟㄟ，視窗操作和標示有什麼關係？為什麼擺在這裡說明？ㄚ，是因為這兩個單元內容都不多，沒地方擺，所以就將就湊在一起的啦！亂點鴛鴦譜就請各位見諒。**

7.1 標示指令

v	**小寫** v**，這是屬於字元標示（**character visual**），按下** v **後您就可以移動游標，游標走過的地方就會標示起來。再按一次** v **或按** Esc **鍵就會結束** v-mode**。**
V	**大寫** V**，這是行標示（**line visual**），按下** V **後會整行標示起來（包括行首前空白的部分），您移動上下鍵，會標示多行。再按一次** V **或** Esc **鍵就會結束** v-mode**。**
Ctrl+v	**這是區塊標示（**blockwise visual**），可縱向標示矩形區域。再按一次** Ctrl+v **就會結束** v-mode**。結束** v-mode **的方式亦可使用** Esc **鍵，或統一使用** Ctrl+c**。**Windows **系統下** Ctrl+v **是複製鍵，可以使用** Ctrl+Q **來替代。**
d	**刪除標示區內容。**
y	**複製標示區內容。**
c	**替換標示區內容。** **標示區內容大小寫互換。**
gq	**重排標示區內容。ㄟ…… 是要先標示好才按的。**”ay **還能不能用呢？當然可以，這樣就會把標示區內容存於** a **緩衝區中。可以用** ”ap **來貼上。**
Shift+>	**標示區內容向右移一個** Tab**。**
Shift+<	**標示區內容向左移一個** Tab**。**

您想更深入嗎？:h visual.txt **就有詳細的介紹。還有，別忘了有** Tab **補全鍵可以用。**

7.2 視窗操作

Ctrl+w n **即** :new**。開一空的新視窗。**

這在 *Vim* **會開在原視窗上半方，也就是視窗一分為二。在** elvis GUI **界面的話，則是實實在在的另開一個新視窗（可不是另啟動一個** elvis **喔！），當然** elvis **的** console **上也是和** *Vim* **一樣，視窗一分為二。而且在** elvis **下，要放開** Ctrl+w **後才按** n**，否則** elvis **會不鳥您的。凡是視窗操作的按鍵都是由** Ctrl+w **來起頭的，**w **就是** window **的意思。**

Ctrl+w s **即** :sp(lit)**，會開一新視窗，且原檔分屬兩個視窗。**

Ctrl+w f **開一新視窗，並編輯游標所在處之** word **為檔名的檔案。**

Ctrl+w q **即** :q **結束分割出來的視窗。**

Ctrl+w o **即** :only! **使游標所在之視窗，成為目前唯一顯示的視窗其它視窗會隱藏起來。**

Ctrl+w j **移至下視窗。**

Ctrl+w k **移至上視窗。還記得** hjkl **的按鍵移動方式嗎？**

:sp *檔名* **開另一新視窗來編輯檔案。**

如果您覺得這樣分割個視窗來編輯檔案不怎麼方便，那您可以利用 X **下的虛擬桌面，多開個** rxvt + *Vim* **來做多檔編輯也是可以的啦！不過這樣不僅會多佔記憶體，而且** *Vim* **中好用的書籤功能就沒法使用了。**

第 8 章

shell 命令及求助系統

8.1 shell 命令

:!外部指令	**執行外部指令。當然您的指令要在** $PATH **環境變數內。**
:!!	**執行前一次執行之外部指令。在這裡的** ! **可不是強迫中止喔！**
@:	**這是** *Vim* **的一個很特殊的指令，可以重複前一次的冒號命令。**
:sh(ell)	**執行** shell**。使用** exit **回到** vim**。**

這在 *Vim* GUI **會在原視窗內顯示執行結果，在** elvis GUI **則會另開一** xterm **視窗。**console **或** xterm **下的** vim **當然就是跳出** vim **進入** shell **中。至於是用哪一種** shell **是可以另外設定的，可由** :set shell= **來設定。**

另外，在此補充一下補全的功能。不是只有 Tab **鍵的補全功能喔！也可以使用上下方向鍵，叫出歷史指令，叫出歷史指令可用於冒號命令及尋找命令（/）。例如，您前已下了** :!ps aux **這個指令，您可以按** : **後就直接按向上方向鍵。別忘了！尋找指令也是可以這樣用喔！**

:r !commond	**這個就妙了！會在游標所在處次一行插入外部指令** commond **執行後的輸出內容。例如** :r !date **就會插入日期時間。這在** elvis **是會插入在游標所在處那一行。**
:n,mw !commond	**以** n **至** m **行內之資料，當做外部指令** commond **的** input**。這算是相當高級的用法了，初學者大概還用不上，不過印象中留有一個這樣的功能，以後總是會用得上的。**
K	**大寫** K **會顯示游標所在處之** word **的** man page **系統線上使用手冊。**

8.2 求助系統

原始 vi 是沒有 on-line help 的，但 *Vim* 及 elvis 則有相當豐富的說明系統。*Vim* 沿用傳統 tag 的方式來找主題，而 elvis 就高明了，是使用 HTML 的格式。原來 elvis 是可以直接閱讀 HTML 檔的，當然不能顯示圖檔，但會標明圖檔的名稱。

用過 pe2(3) 嗎？好像 F1 是救助鍵已讓大家公認。*Vim* 預設按 F1 就會叫出說明檔。elvis 當然沒有，不過您可以自行設定，在 `/.elvislib/elvis.rc` 中加上

```
map #1 :helpˆM
```

就可以了。須注意的是 ˆM 是按 Ctrl+v 後不放再按 M 或 Enter 鍵，表示馬上執行。ㄟ，ˆM 是特殊字元，是一個字元，而不是兩個字元，您移動游標在 ˆM 上就知道了，不是我文中的，文中的是直接打出來的兩字元 ˆM。在 console 下要打特殊字元的話，要 Ctrl+Shift+v 然後按您要的鍵。

`:h name` 這樣就會叫出 name 這個說明檔，如果後面沒接 name， 則會叫出 help.txt 這個總說明檔（在 evlis 是 elvis.html） h 是 help 的縮寫。ㄟ，name 記不清楚時，別忘了 Tab 補全。 在 *Vim* 的說明檔中，遇有兩個 |圍住的主題，把游標移到上面 就可以使用 Ctrl+] 來叫出這個主題的說明，Ctrl+t 可以回到原來的地方。elvis 中也是可以這樣用。`:q` 可結束說明檔，回到原編輯檔案。

`:ver` 會顯示版本、編譯資訊，編譯時加入之參數也會顯示出來， 讓您知道有加入些什麼功能，因為有些功能在編譯時就得加入。 其中正號 + 表示有此功能，減號 –表示無此功能。elvis/nvi 只能顯示版本資訊，無法顯示編譯資訊。[1]

在此補充說明一點，各位有沒有覺得 `:q` 很常用到，好了！就依 pe2 的習慣設為 F4 吧！怎麼設？找上面叫出說明檔怎麼設，您就依樣畫葫蘆就可以了！*Vim* 的設定檔在 `/.vimrc`。個人是直接設成 `:q!` 以免麻煩，但要記得存檔喔！

[1] 個人都會編譯出一個全功能的 `vim` 來備用，而且 GUI 及 console 下使用的各編一個，以加快 console 下的啟動速度。

第 9 章

set 功能設定

本單元可說是 *Vim* 的微調功能，可依您個人的喜好做有限度的調整。由於 *Vim* 做了相當多的擴充，文內主要是述敘 *Vim* 的設定，但 elvis **沒有的也會標明。但並不打算一網打盡，只說明重要、常用的部份，其它的可以** `:h option-list` **來查閱。**

在系統上，會有個 `vimrc_example.vim` **檔，可以依據這個檔案來設定一些常要用到的內容。**

9.1 該在何處設定呢？

9.1.1 *Vim* 的設定檔

可在線上做設定，例如 `:set ai` **或** `:set noai`**，**`ai` **是** autoindent **的縮寫，這樣就可以馬上改變縮格的設定，但離開** `vim` **後就又恢復原狀。要永久設定就得設在設定檔中。** `vim` **的設定檔在：**

```
~/.exrc
~/.vimrc
~/.gvimrc     GUI 版本
$VIM/vimrc    系統預設值，最好不去修改
$VIM/gvimrc   GUI 版本
$VIM\_vimrc   Windows 版本
$VIM\_gvimrc
```

您要把 GUI **的設定設在** `$HOME/.vimrc` **也是可以的，但分開來可能對以後要修改時會比較找得到地方改。那** `$VIM` **在哪裡呢？**`/usr/share/vim` **或** `/usr/local/share/vim` **這是編譯時就決定的，但可在** `$HOME/.bash_profie` **或** `$HOME/.bashrc` **中另設。**Windows **版本的** `$VIM` **則在** `C:\Vim` **這個目錄中，亦可設在** autoexec.bat **中。**

9.1.2 elvis 的設定檔

```
~/.exrc
~/.elvislib/elvis.rc
C:\Program Files\elvis\elvis.rc  Windows 版本
```

如果您的系統上同時有 *Vim* 及 elvis，則使用 $HOME/.exrc 要小心，以免互相影響了設定。好處是可以把共同的設定設在 $HOME/.exrc 裡。提醒您一下！設在設定檔中時 set 前當然是不必冒號的。

9.1.3 nvi-m17n 的設定檔

```
~/.exrc
~/.nexrc
```

9.2 如何得知目前的設定

```
:set 或 :se     會顯示所有經過修改的部份，就是和預設值不一樣的部份。
:set all        顯示目前所有設定值內容。
:scriptnames    顯示各種設定檔的所在路徑及其檔名。
:set option?    顯示 option 這設定的目前值。
:option         直接線上設定，有些設定需加 = 後加上設定值內容。
:set nooption   取消該設定。:set 後面是可以多重設定的。例如
:set autoindent noconfirm autowrite，這樣三種設定就會同時重設。
```

您當然可以改設定檔來改變設定值。在 *Vim* 也可以使用 :opt[ion] 來直接線上設定，會列出目前的設定，在 set 這個字上按 Enter 即可改變設定，或就直接修改其值亦可，改完後按 :q 就可以了。在簡短說明處按 Enter 則會叫出該部份的說明檔給您參考，您說方不方便？改好後

```
:mk[exrc]     則會寫入 ~/.exrc 檔
:mkv[imrc]    則會寫入 ~/.vimrc 檔
```

當然您得先搞清楚您目前所在目錄在什麼地方，如果您是在家目錄啟動的那就不用擔心了，否則找不到您的新 .vimrc 可不要寫信來罵我。:-) elvis **可就沒這麼方便了，得自行修改 $HOME/.exrc 或 $HOME/.elvislib/elvis.rc。**

9.3 各種 set 功能說明

方括號內的是縮寫，由於 *Vim* 新增的指令、設定項實在是相當多，建議一開始使用完整的寫法，等熟悉後再來使用縮寫會比較好。

autoindent[ai]

自動縮排，也就是說如果本行是從第五個字元開始寫的，您按 Enter **後游標就會停在次行第五個字元處。預設是不打開的。**

autowrite[aw]

檔案一有更動就會自動存檔。預設不打開。

background[bg] （*Vim* 才有）

可設成 dark **或** light**，這是兩種不同的** highlight **顏色設定，系統預設的內容是在一個** `synload.vim` **檔中，詳見** `$VIMRUNTIME/syntax/synload.vim`**。不過您要更動顏色的設定，最好是設在** `$HOME/.vimrc` **或** `$HOME/.gvimrc` **中，原始檔預設檔盡量不要去動她。**

ㄟㄟㄟ，你從沒提過 `$VIMRUNTIME` **好不好！其實這是最近版本的 *Vim* 為了不至安裝新版本時把舊版本的一些設定或** macro **檔幹掉，所以** `$VIMRUNTIME` **就是** `$VIM/vimxx`**，**xx **就是版本號碼啦！例如您使用的是 *Vim* 6.1 版，那麼就是** `$VIM/vim61`**，往後安裝** 6.2 **版時，會把** runtime files **裝在** `$VIM/vim62`**，這樣就不會把舊的** runtime files **給蓋掉。**

backup[bk]

是否要 backup file**。預設不打開。**

writebackup[wb] （*Vim* 才有）

在寫入檔案前先備份一份，和 backup **的作用不盡相同，請** `:h backup-table`**。預設是打開的，所以您如果不要** backup**，那要關掉的是這個項目，而不是** backup**。但請先檢查一下您編譯時是不是有編譯進去，請** `:ver`**。**

backupdir[bdir] （*Vim* 才有）

設定存放 backup file **的目錄。預設在所編輯的檔案之所在目錄。**

binary[bin] （*Vim* 才有）

設在編輯二進位檔狀態，這是防止存二進位檔時把 EOL **也寫進二進位檔，那就會悔不當初，如果是圖檔將會無法再觀看，如果是可執行檔就無法執行了！因此預設是** off。elvis **會自動判斷是否為二進位檔，而且會分成左右兩半，左半部會以** 16 **進位的方式顯示，右半部則是以** ASCII **的方式來顯示。**

browsedir[bsdir] （*Vim* 才有）

瀏覽檔案的目錄，GUI **版本始有。預設是上一次瀏覽的目錄。就是** GUI **版本功能表上的** [File] ➪ [Open] **會打開的目錄。**

cindent[cin] （*Vim* 才有）

寫 C **時很好用，**indent **比一般敏感，專為** C **程式碼而設。預設** off。**編輯** C/C++ code **時會自動打開。**

cmdheight[ch] （*Vim* 才有）

狀態列的行數，預設一行，建議設成兩行。

compatible[cp] （*Vim* 才有）

設為和原始 vi **相容的狀態，**vim **的擴充功能會被抑制。預設** off。

confirm[cf] （*Vim* 才有）

各種確認動作。預設 off。

directory[dir]

swap **檔存放的目錄。前面單元已有說明。**

fileformat[ff] （*Vim* 才有）

這是寫入檔案時置放 EOL(end of line) **的形式**

```
dos 是以 0x0D0A 來斷行。
unix 是以 0x0A 來斷行。
mac 是以 0x0D 來斷行。
```

預設以各系統平檯而定，在 Linux **當然是** unix **形式。**

fileformats[ffs] （*Vim* 才有）

可指定多個，會依載入的檔案形式來調整 ff**。**

例如 `:set ffs=unix,dos ff=unix`

則預設為 unix **格式，但如讀入的是** dos **格式的檔案，會自動調整為** dos **格式，這樣存檔時就會以** dos **格式存檔（狀態列會顯示）。。此時如要改成** unix **格式，可** `:set ff=unix` **然後存檔就會轉成** unix **格式，反之亦然。**

如果不這樣設，也就是您不管 ff **或** ffs **都設成** unix**，那讀入** dos **格式的檔案時在每行尾會出現** ^M **這個字元（就是** 0x0D **啦！）這時縱使** `:set ff=unix` **也來不及了！只好** `:%s/^M//g` **來消去這個** ^M**。ㄟ，還記得怎麼替換嗎？就是把** ^M **換成沒有啦！而且** ^M **怎麼打出來的還記得吧！翻一翻前面的單元吧！**

Hey**，你怎麼知道是** 0x0D **呀！好吧！告訴您一個密秘，您把游標移到** ^M **那個位置，然後按** ga **在狀態列就會顯示** 10，16，8 **進位的值。其它的字元也是可以如此顯示。**a **就是** ascii **的意思。但這是** *Vim* **的擴充功能，**elvis **沒有。**elvis **縱使載入** dos **格式的檔案，也是會自動把** ^M **隱藏起來。**

ignorecase[ic]

尋找時不分大小寫，這對中文會造成困擾。預設 off**。**

incsearch[is] （*Vim* 才有）

加強式尋找功能，在鍵入 patern **時會立即反應移動至目前鍵入之** patern **上。預設** off**。**

hlsearch[hls] （*Vim* 才有）

尋找時，符合字串會反白表示。預設 off**。如果您是使用** `vim` **的預設的** `vimrc` **檔的話，可設在** F8 **鍵來切換。**

textwidth[tw]

是一種 word wrap **的功能，從左起算之固定每行的最大字元寬度。超過此寬度就會自動折行，這可是真的折行，也就是說在折行處會插入** EOL**。預設是** 0**，也就是沒有** word wrap **的功能。**

wrapmargin[wm]

和 textwidth **作用相同，只是是從右視窗邊向左算起要幾個字元起折行。預設是** 0**。**

wrap

這也是折行功能，可是只是螢幕效果的折行，實際上並沒有插入 EOL。

wrapscan[ws]

這和折行沒有關係，是指尋找時，找至檔尾時，是否要從檔首繼續找。預設是要。

paste （*Vim* 才有）

這是防止在做剪貼時位置會不正確，前面單元已有說明。

ruler[ru] （*Vim* 才有）

會在狀態列顯示游標所在處之行列狀態，預設不打開，但建議打開。最右邊之代號的意義如下：

`Top` **檔案第一行在螢幕可見範圍。**
`Bot` **檔案最後一行在螢幕可見範圍。**
`All` **檔案首尾皆在一個螢幕範圍內。**

如非以上三種情形，則會顯示相對百分比位置。

statusline[stl] （*Vim* 才有）

狀態列顯示的格式，使用預設就可以了，如果您想騷包一下的話，那就請您 `:h stl`**。**

shiftwidth[sw]

指由 >>**移動整行內容時，一次移動的字元寬度，一般是使用** Tab **的值，但可由這個設定來改變。**

tabstop[ts]

一個 Tab **鍵寬度。預設是** 8 **個字元寬度。最好不要隨便改，以免您寫的東西由其它編輯器來閱讀時造成困擾，為解決這個問題，** *Vim* **另有種** softtabstop **及** modeline **的機制，在以下各節會詳細說明。**

showcmd[sc]

在狀態列顯示目前所執行的指令，未完成的指令片段亦會顯示出來。

showmode[smd]

在狀態列顯示目前的模式，例如是 Insert mode **或是** Visual mode**。當然平常的** normal mode(commond mode)**是不顯示的。在載入檔案的同時，會在這個地方顯示檔案名稱及其總行數、總字元數。**

viusalbell[vb]（*Vim* 才有）

以螢幕閃動代替 beep **聲。**

number[nu]

顯示行號。注意，冒號命令也有 :nu **這是顯示游標所在行的行號，您嫌多打一個字的話，**:# **也行。不過如果** ruler **打開的話，在狀態列本就會顯示門前游標所在處的行列值。**

list

這也可以算是一種模式，list mode**。就是** Tab **的地方會以** ^I **顯示，而行尾之** EOL **會顯示成** $**。可以讓您清楚的知道** Tab **在哪裡，折行是不是真的。**

swapfile[swf]（*Vim* 才有）

是否需 swap **至磁碟。如果設為** noswf **的話，那將不會有** swapfile **產生，通通會載入在記憶體中。預設是要** swapfile**。**

fileencoding[fe]（*Vim* 才有）

首先，先鼓掌一下，啪啪啪……，因為有支援 Taiwan**，也支援** XIM**，也就是說可以使用** xcin-2.5x **來作輸入，當然您用** xcin-2.3x **配合** XA **也是可以啦！但前提是您要把** multi_byte **編譯進去，這在一開始就講過了。預設是使用** ansi**。**set guifont **及** set guifontset **已在一開始講過，在此就不重複了。**

history[hi]

記錄冒號命令的歷史紀錄檔，就是可以用上下方向鍵叫出來的那鍋。預設是 20 **筆。**

9.4 關於 softtabstop[sts]

幾乎所有的 OS **及軟體都設定** Tab **就是** 8 **個字元長，這已經是個公認值，您硬要去改變它的話恐怕帶來許多不便，但實際上關於程式風格，許多人又認為** 8 **個字元太長了，幾個巢**

狀迴圈下來就需折行，反而不方便。因此 *Vim* 體貼您，內建了 softtabstop 的功能，就是由 *Vim* 來代您製造出一個假的 Tab，實際上是空白字元組成的 Tab。

舉個例子來說明比較清楚。

```
set softtabstop=4
set shiftwidth=4
```

這樣會由 4 個空白字元取代一個 Tab，您按 Tab 鍵 `vim` 就跳 4 格，需注意的是，如果您按了三次 Tab 鍵，那就是一個實際的 Tab 加上四個空白字元，可不是 12 個空白字元喔！是混合 Tab 及 Space 的。

問題來了！那我要按真正的 8 字元的 Tab 時怎麼辦？簡單，還記得怎麼按特殊字元嗎？Ctrl+v Tab 或 Ctrl+v I 就可以了，那就是如假包換的 8 字元長之 Tab。當然，您按兩次 Tab 不就得了！:-)

另一個更大的問題也來了！在 Un*x 系統的有些設定檔，例如 Makefile, snedmail 的設定檔，他們會認真正的 Tab，因此如果您有程式開發或系統管理的需要的話，最好是不要隨意更動 Tab 值，也不要使用 softtabstop，可以使用 *Vim* 獨特的 modeline 來控制，請繼續研讀下一節。

9.5 *Vim* 的 modeline

這個是什麼哇？簡單的說，*Vim* 有能力去解讀所編輯檔案內的 *Vim* 控制參數，而這些控制參數是寫在檔案的註解行裡頭的，例如 sh script 的註解行是以 # 來開頭的，那 *Vim* 的控制參數就可以寫在這些註解裡頭，*Vim* 在開這個檔的同時，也會去控制自己的一些行為，當然，包括了上一節所說的 Tab 的長短，而 shell 本身則會忽略這些註解。

例如，現在把一個 sh script 檔裡頭，在檔案最後一行，設了個：

```
# vim:  ts=2
```

這樣一來，按個 Tab，實際上顯示出來會只有兩個空格，別擔心仍然是正港 Tab ，不相信的話，可用另一種編輯器打開就知道了。這個好用吧！詳細請進入 `vim` 後 `:h modeline`。

當然，這個功能很方便，但並不是全然沒有限制的，以下有些限制要注意：

1. 控制 modeline 本身的設定，不能寫在檔案裡頭，vim 會來不及讀他。例如：在 sh script 檔案中設定 # vim: set modelines=0 這是行不通的。
2. 這些設定必須要在檔案的開頭五行，或結尾五行內設定，否則 vim 會忽略。
3. 註解符號後要至少一個空白(sapce 或 Tab 皆可)隔開。例如：

```
/* vim:noai:sw=3 ts=4 */  => C 程式碼 /* 和 vim 間至少要有個空白
// vim:ts=2               => C++
# vim:ts=2               => sh script, perl, python, tcl
" vim:ts=2               => vim script
```

9.6 關於折行

前面已說過 set wrap 就可以造成螢幕折行，可是卻會把一個英文單字折成兩半，實在很不雅觀。好了，*Vim* 再體貼您一次，:set linebreak(lbr) 就會避免這種問題發生，會在空白或標點符號的地方來折行，但也仍屬螢幕折行，並不會插入 EOL。

在一般的文書排版處理，甚至是寫程式碼，設定系統的設定檔，這些時機，可能自動折行並不很適合(指會插入 EOL 的)，很容易把寫好的內容，因為一個不小心就都弄亂了，因為程式一般是一行一行在讀的，把一整行分成兩半，所讀的結果就不一樣了，除非加入程式碼的折行標誌，例如一般的倒斜線[1] \ 或 TEX/LATEX 的百分號 %。而在排版軟體如 TEX/LATEX，插入 EOL 的折行，會造成中文字間多出個空白，這在 HTML 格式的檔案亦同，因此，在做折行的決定時，也請考慮一下其他的『副作用』。

9.7 我的設定檔

以下是我個人的設定檔，僅供參考用：

9.7.1 .vimrc 範例

```
" ~/.vimrc by Edward G.J. Lee
" 這個記號就是 vim 的註解，通常情形 vim 會忽略。
" This code is Public Domain
"
set backspace=indent,eol,start

set autoindent     " always set autoindenting on
```

[1] *Vim* script 的折行標誌也是倒斜線，但這個倒斜線不是像一般 script 一樣放在本行的最後，而是放在次一行的開頭處，請小心不要放錯了。

```
set history=50     " keep 50 lines of command line history
set ruler          " show the cursor position all the time
set showcmd        " display incomplete commands

set enc=big5       " 檔案編碼設成 big5

"使用 :update 代替 :w，以便在有修改時才會存檔，注意，這和 vi 不相容。
map <F2> :up<CR>
map <F3> :up<CR>:q<CR>    " 存檔後離開
map <F4> :q!<CR>    " 不存檔離開
map <F5> :bp<CR>    " 前一個 buffer 檔案
map <F6> :bn<CR>    " 下一個 buffer 檔案
" 單鍵 <F7> 控制 syntax on/off。倒斜線是 Vim script 的折行標誌
" 按一次 <F7> 是 on 的話，再按一次則是 off，再按一次又是 on。
" 原因是有時候顏色太多會妨礙閱讀。
map <F7> :if exists("syntax_on") <BAR>
     \   syntax off <BAR><CR>
     \ else <BAR>
     \   syntax enable <BAR>
     \ endif <CR>
" 按 F8 會在 searching highlight 及非 highlight 間切換
map <F8> :set hls!<BAR>set hls?<CR>
" Toggle on/off paste mode
map <F9> :set paste!<BAr>set paste?<CR>
set pastetoggle=<F9>

map <F10> <ESC>:read !date<CR>    " 插入日期
map <F11> :%!xxd<CR>       " 呼叫 xxd 做 16 進位顯示
map <F12> :%!xxd -r<CR>    " 回復正常顯示

"Default backspace like normal
set bs=2

"Display a status-bar.
"set laststatus=2

"Show matching parenthese.
set showmatch

" 以下是顏色設定，詳細請 :h hi
hi Comment      term=bold ctermfg=darkcyan
hi Constant     term=underline ctermfg=Red
hi Special      term=bold ctermfg=Magenta
hi Identifier   term=underline ctermfg=cyan
hi Statement    term=bold ctermfg=Brown
```

```
hi PreProc      term=bold ctermfg=DarkYellow
hi Type         term=bold ctermfg=DarkGreen
hi Ignore       ctermfg=white
hi Error        term=reverse ctermbg=Red ctermfg=White
hi Todo         term=standout ctermbg=Yellow ctermfg=Red
hi Search       term=standout ctermbg=Yellow ctermfg=Black
hi ErrorMsg     term=reverse ctermbg=Red ctermfg=White
hi StatusLine   ctermfg=darkblue  ctermbg=gray
hi StatusLineNC ctermfg=brown   ctermbg=darkblue

set viminfo='20,\"50    " read/write a .viminfo file, don't store more
                        " than 50 lines of registers

" 方便中文重排設定
set formatoptions=mtcql

" 設定 modeline
" vim: ts=2:
```

9.7.2 .gvimrc 範例

```
" ~/.gvimrc by Edward G.J. Lee
" This code is Public Domain
"
set mousehide           " 當輸入文字時隱藏 mouse
map <S-Insert> <MiddleMouse>
map! <S-Insert> <MiddleMouse>

" Only do this for Vim version 5.0 and later.
if version >= 500
  let c_comment_strings=1
  if !exists("syntax_on")
    syntax on
  endif

  hi Comment      guifg=DarkBlue
  hi Constant     gui=bold guifg=Magenta guibg=#fffff5f2eed8
  hi Special      guifg=Blue guibg=#fffff5f2eed8
  hi Identifier   gui=bold guifg=DarkCyan
  hi Statement    gui=bold guifg=brown
  hi PreProc      gui=bold guifg=black
  hi Type         guifg=darkgreen
  hi Ignore       guifg=bg
  hi Error        guibg=Red gui=bold guifg=White
```

```
  hi Todo         guifg=Blue guibg=Yellow
  hi ErrorMsg     guibg=Red gui=bold guifg=White
  hi Normal       guibg=#fffff5f2eed8
  hi ModeMsg      guifg=DarkBlue guibg=yellow
  hi StatusLine   gui=bold guifg=lightgray guibg=DarkBlue
  hi StatusLineNC gui=bold guifg=lightgray guibg=brown
  hi Cursor guibg=green guifg=NONE
  hi lCursor guibg=Cyan guifg=NONE

endif

" 讓 ex(:) mode 時可以輸入中文(gvim)
set imcmdline
" 讓 xcin 在 insert mode 不自動出現(gvim)
set imdisable
```

9.7.3 .vim 目錄範例

其實，一些設定檔除了家目錄下的 .vimrc 及 .gvimrc 外，可以置放於家目錄下的一個子目錄 .vim，其目錄結構可以仿 $VIMRUNTIME 這個系統上的結構。例如，我的 .vim 結構如下：

```
edt1023:~$ tree .vim
.vim
|-- filetype.vim
|-- plugin
|   |-- format.vim
|   `-- tex.vim
|-- syntax
|   |-- lily.vim
|   `-- svg.vim
`-- view
    `-- ~=+bg5.txt=

    3 directories, 6 files
```

其中，filetype.vim 會在系統的 filetype.vim 載入後蓋過去，也就是家目錄的會先優使用。format.vim 及 tex.vim 是進入 vim 馬上會載入的 *Vim* script，主要用於中文重排及 TEX/LATEX 排版系統編輯時的方便指令、按鍵。lily.vim 則是 GNU LilyPond 這個樂譜排版系統檔案的語法顏色定義檔，svg.vim 則是 SVG 向量圖檔的語法顏色定檔加強版。那 view 那個子目錄下的又是什麼呢？這是下 :mkview 指令時所存的一些狀態檔，在 :loadview 要參考到，請參考第 11 章，第 11.1.3 小節，頁 52，這個檔是個隱藏檔。

第 10 章

規則表示式的運用

在本系列文章一開始就說明了學 *Vim* 可以順便學規則表示式（regular expression**，以下簡稱** regexp**），那為什麼到現在才來講呢？因為** regexp **說簡單也算不很難，但您要深入去使用的話，有時會馬上看不出一個複雜的** regexp **在說些什麼的，就曾有人形容** regexp **為「有字天書」！而且在 *Vim* 整體都還沒一個概念就加入** regexp **的話，那後面的單元恐怕就沒人看了！而** regexp **各家有各家的** extensions**，這也是大家視為畏途的原因之一，不過總是大同小異，只需注意一下就可以了。**

目前先不必管別家怎麼說，就讓 *Vim* 暫時先成為我們的「標準」，以後碰到其它程式的 regexp **應該就可以觸類旁通。以下我們盡量由實例去瞭解。當然，小小的一篇文章是沒有辦法詳盡介紹，只能撿重點來說明了。如有疑問，可** `:h pattern` **或在** Un*x **系統中可** `man 7 regex`**，甚至** `man ed`**，**`man sed`**，**`man grep`**，**`man awk`**，**`man perlre` **裡面也是會說些** regexp**，但要注意和 *Vim* 差異的地方！其中** perl **的** regexp **應該是最完整的了，如果您的系統沒有** perl **那應該是「稀有動物」了！**:-) **ㄟㄟㄟ！*Vim* 只是一個編輯器，可不是獨立的程式語言(雖然，內建的 *Vim* script 是有點好用)！**

10.1 基本的匹配

* **指前所綁住的字元或字元集合，出現** 0 **次或** 0 **次以上。**

\+ **和** * **作用相同，但不包括出現** 0 **次。**

\= **指前所綁住的字元恰好出現** 0 **或** 1 **次。**

\| **這是多選，就是** or **的意思，被** \| **隔開的** pattern**，任一個符合的話就算符合。**

\+, \=, \| **會加上一個** **，是因該字元在 *Vim* 就具有特殊意義，在一般的** regexp **中是** +, ?, | **就可以了，只是提醒您一下，以免搞混了！ 記住** \+ **是不可數的！用辭不是很精確，只是幫助您記憶啦！在** elvis **及** ed **中是使用** \? **來匹配出現** 0 **或** 1 **次，而不是** \=**，這裡要非**

常小心！

[實例] `dg*`
指 * 前所綁住的字元 g 出現 0 次或 0 次以上。也就是說 d(出現 0 次)，dg, dgggg, dgggggggg 都是符合這個 pattern。如果您下尋找指令 `/dg*`，那符合這個 pattern 的字串都會被找出來。如果用在代換就要非常小心了，像 extended 中的 d 也是會被置換掉的。例如您下 `:%s/dg*/test/g` 的話，那 extended 這個字會換成 extentestetest。

shell 中使用的通用字元為 pattern matching notation 和 regexp 是不同的意思。dg* 在 shell 中是解為以 dg 開頭的任意字串，這就不包括 d 在內了，也就是說在 shell 中，* 是代表任一字元或字串，這在初接觸的朋友很容易就搞混，請千萬小心。

[實例] `dg\+`
dg, dgg, dgggggg 皆符合，但 d 則不符合。如果是 dg\= 的話，就只有 d、dg 這兩個符合了。

[實例] `:%s/The\|All/test/g`
全文中只要是 The 或 All 都會被替換成 test。注意，如果文中有 There 也是會被替換成 testre！要如何避免這種情形呢？下面會另述及限定使用法。

[實例] `/123-\=4567`
這樣會找出，123–4567 及 1234567。當然 123–456789 也是會被找出來。

`[...]` 字元集合，表示中括號中所有字元中的其中一個。

`[^...]` 這是上述 [...] 的補集，表非中括號內字元的其中一個。

. 除換行字元外的任一單一字元。指本身，非指前所綁之字元。就好像 shell 中的 ? 一樣。如果要指定真正的英文句點，要用 \來 escape，就是說 \. 這時的 . 是代表真正句點，而不是 regexp 中的特殊意義。其他如 * 亦同。

[實例]
`[Aa]`
A 或 a 其中的一個。
`[12345]`
12345 其中的一個數目字。可用 [1–5] 來表示。連續性的數目字或字元可用 –來隔開，寫出頭尾來代表就可以了。[0–9] 就表 0 到 9 的數目字，[a–d] 就代表 abcd 四個英文字母

[實例] `W[0-9]*\.cc`
這個例子是說以 W 開頭，後接 0–9 其中一個或多個數目字或不接什麼，然後是一個句點，最後是 cc。所以 W.cc，W1.cc，W2.cc，W345.cc，W8976543287.cc 皆符合。如果要表示 W 及 .cc 間夾一個以上的數目字，要寫成 `W[0-9][0-9]*\.cc`。

[實例] .*
這代表任意字元或字串，或什麼都沒有，腦筋急轉彎，對照前面的定義想一下。當然這是不包括換行字元的。

[實例]
`[^M]` 表除 M 以外的任意字元。
`[^Tt]` 表 T 及 t 以外的任意字元。
`[^0-9]` 表非數目字之字元。
`[^a-zA-Z]` 表非英文字母之字元。

注意，^ 要在中括號內，且在最開頭的地方，否則另有含意。

^ 匹配行首，指其後綁住的字串，出現在行首才符合。
$ 匹配行尾，指其前綁住的字串，出現在行尾才符合。含換行字元。

不是在行首的 ^ 指的是 ^ 這個字元。不是在行尾的 $ 是指 $ 本身這個字元。

[實例] /^What
這樣只有在行首的 What 才會被找出來。注意！ Whatever, What's 也是會被找出來。如果是 /What$ 則是在行尾的 What 才會被找出來。

[實例] ^$
這是什麼東東？行首也是行尾的行。丫，就是空白行嘛！當然也不能說這個行是沒有什麼東東啦！空白行至少也是會有個換行字元。在後面會詳述如何消除全文的空白行。

`\(...\)` 記憶 pattern，可由 \1, \2... \9 來叫出。

[實例] `:%s/\([a-z]\)\1/test/g`
這樣 aa, bb, cc, dd,... zz 都會被 test 替換掉。這和 `:%s/[a--z][a--z]/test/g` 是不一樣的意思，後者會把 aa, ab, ac... ba, bb, bc... zz 都換成 test。也就是說 `\(...\)` 由 `\1` 叫出時會有對稱性的配對出現。

[實例] :%s/\(.\)\(.\)r\2\1/test/g
會將中間為 r，前有二個任一字元，後有兩個具對稱性的字元所組成的字串替換成 test。\2 是呼叫第二組 \(.\)，而 \1 是呼叫第一組 \(.\)。例如：12r21，cfrfc，7grg7 等都會被替換成 test。

\< 匹配字（word）首。所謂 word 包括文數字及底線。

\> 匹配字尾。這就是前所提及的限定用法，被 \<，或 \>括住的 pattern 就會被限制住，使 regexp 不能再向右（左）擴充解釋。

ed 及 perl 中可以 >b 來表示這兩個符號，perl 中只支援 >b，ed 則 >b 及 \<, \>皆支援。但在 perl 可多加個 ? 來限制 regexp 的擴充解譯。功能上而言，這是和 ^ $ 一樣的定位樣式（anchor pattern）指所綁住的字串必須是單字邊界（word boundary），前或後或前後除了空白字元及標點符號外不可再有其它字元。在 *Vim* 中 \b 是表示 <BS>即 Backspace 鍵。

[實例] :%s/\<abbbc\>/test/g
這樣只有 abbbc 才會被替換成 test。如果沒有這樣限定，:%s/abbbc/test/g，那 deabbbcly 中的 “abbbc” 亦會被替換成 test。所以前面 :%s/The\<All/test/g 可換成 :%s/\<The\>\|\<All\>/test/g 這樣一來，There 就不會被替換成 testre 了！

[實例] :%s/\<abbbc/test/g 這樣的話，只要是以 abbbc 為首的字(word)，其中的 abbbc 的部份都會被 test 所替換。注意！是指字首，而不是指行首。所以 abbbc，abbbcerd ，abbbckijuds 都符合。

\{n,m} 指前所綁住的字元或字元集合最少出現 n 次，最多出現 m 次。

這在一般的 regexp 表示成 \{n,m\}。*Vim* 及 elvis 兩種表示法皆支援。perl 則直接使用 {}。以下會舉四種不同的例子，請大家發揮一下想像力。:-)

[實例] \{最小值，最大值}
如 [0–9]\{3,4} 匹配至少三位數，但不可多於四位數的數目字。如：
123
12
1
123456
1234567
12345678
1234
12345

如果下 `:%s/[0-9]\{3,4}/test/g` **的話，那** 1，12 **這兩組不會被替換，因為不滿** 3 **位數。而** 12345，**則會換成** test5。123456，**則會換成** test56。12345678 ，**則會換成** testtest。1234567 **也是會換成** testtest。123，1234 **這兩組則會被替換成** test。**您可以親自操作一次就知道怎麼一回事了。操作時最後加** gc **來** confirm，**這樣您會更瞭解實際替換的內容。ㄟ，別忘了** u **可以回複您的編輯動作。**

[**實例**] \{**數目字**}
xy\{20} **表示** x **後接** 20 **個** y。 e[x-z]\{4} **表示** e **後接有四個字元，是** x,y,z **的其中一個的組合。如：**exxxx, exyyz, ezzyz, exyzz **皆符合。**

[**實例**] \{**最小值，**}
xy\{2,} **表** x **後接至少二個的** y。**相當於** xyyy* **或** xyy\+。

[**實例**] \{**，最大值**}
xy\{,4} **表** x **後接至多四個或更少的** y **（可能沒有）。因此** x, xy, xyy ,xyyy, xyyyy **皆符合。**

10.2 中介字元（metacharacter, or character classes）

主要是簡化 regexp **的書寫。**

- **s** **表空白字元，即** Space **或** Tab。
- **S** **表非空白字元。**
- **d** **表數目字（**digits**），即** [0–9]。
- **D** **表非數目字，即** [^0–9]。
- **w** **表一般字元（**word character**），包括底線。即** [0–9a–zA–Z_]。
- **W** **表非一般字元，即** [^0–9a–zA–Z_]。
- **a** **表英文字母（**alphabetic character**），即** [a–zA–Z]。
- **A** **表非英文字母，即** [^a–zA–Z]。
- **l** **表小寫字母（**lowercase character**），即** [a–z]。
- **L** **表非小寫字母，即** [^a–z]。
- **u** **表大寫字母（**uppercase**），即** [A–Z]。
- **U** **表非大寫字母，即** [^A–Z]。

原始 vi **不支援此種中介字元。使用中介字元的比對速度將會比使用字元集合** [] **的快。**

10.3 全域性的指令

:[range]g/pattern/[cmd]

cmd 是 ed 可用的指令，預設是 p(print)，您可查一下 man ed，就可以知道有什麼指令可用。這個小節裡主要是說明 d(delete) 的功能。因為是要說明如何消除空白行。需注意的是，d 是行刪除指令，凡含 pattern 的整行都會被刪掉，而且 range 不指定的話，預設是全篇文章，因為 g 就是代表 globe。

在 *Vim* 的 help 檔裡說的是 ex 指令，但 ex 實際上是和 *Vim* 連結的，因此這裡特別指出 ed。但 ed 的指令少數可能會和 *Vim* 的 ex 不同，這是因為 ed 和 *Vim* 並非同步在發展，作者也非同一人。

:g/^$/d

這樣就會刪除全文的空白行。前面已提過 ^$ 代表的是空白行。但這裡有個問題，如果空白行裡包含了其它空白字元（即 Space 或 Tab）的話。表面看起來是和一般空白行一模一樣，但卻暗藏玄機，用上面的方法就無法刪除這種空白行了！怎麼辦？來！看招！

```
:g/^[<Space><Tab>]*$/d
```

在 *Vim* 或 elvis 裡您可以如此照打，也就是 <Space>代表空白字元，<Tab>代表按 Tab 鍵的結果。在原始 vi 則不行，得自行按出特殊字元出來，就是 Ctrl+v Space 及 Ctrl+v Tab。或採更簡單的打法：

```
:g/^\s*$/d
```

還記得中介中元嗎？好用吧！少打了不少字。:-) 意思就是刪除含 0 或 1 個以上空白字元的行。

有些書中寫成 :%s/^$//g 可以刪除空白行，這是錯誤的，因為 :s 這 個指令只更動一行裡的內容物，但不會做刪除一行的動作。

10.4 & 替代變數

代表置換時合於 patern 的字元或字串。

[實例] :%s/\d\d\d\d\d\d\d\d\d\>/ID:&/g
這樣全文中的身份證字號前就會加上 ID: 字樣，也就是說 T123456789 會被換成 ID:T123456789。還記得嗎？ \d 就是 [0-9]，\u 代表大寫的英文字母。加個 \>是防止 T12345678999 也被換掉。當然前面再加個 \<更保險。ID: 字樣您用中文也行！另一個好用的例子是電話號碼前加上 TeL:，就請您自行練習了！

[實例] 將檔案 3 至 7 行的資料向右移 2 個空白
```
:3,7s/.*/  &/
```
但這樣連空白行也是會插入空白字元，較高明的做法是：
```
:3,7s/.\+/  &/
```
這樣空白行就不會去動它了！想通了 .* 及 .\+ 的意思了嗎？往前翻一下 . * \+ 的定義。

[實例] 將檔案 3 至 7 行的資料向左移 2 個空白
```
:3,7s/^  //
```
就是刪去行首的二個空白啦！

[實例] 將全文的 Edward 這個單字，前後加上中括號
```
:%s/\<Edward\>/[&]/g
```

[實例] 將全文的 Edward 這個單字，改成大寫的。
```
:%s/\<Edward\>/\U&/g
```

ㄟ！\U 不是代表非大寫字母嗎？喔！您搞錯位置了。\U 在 pattern 的位置的時候是指非大寫字母的樣式，即 [^A–Z]，但如果是在置換字串位置的時候是指將其後的字串通通改成大寫。與其相對的是 \L，會將其後的字串改為小寫。詳細請 :h sub-replace-special。

[實例] 將全文每行最後加上
這個 HTML tag。
```
:%s/.*/&<BR>/g
```

怎麼樣，是否已感覺到 regexp 威力無窮了呢？還是您已經快睡著了呢？:-) 不過也請您想想，如果是在沒有 regexp 功能的編輯器裡，範例中的一些動作您會怎麼做呢？一個一個去改？

10.5 greedy 陷阱

regexp 會有貪心的傾向，什麼意思呢？就是說在同一行內，如果有多個符合 pattern 的情形，會找最長的那一個。請注意！greedy 的特性是針對會反覆比對的 regexp 而言，例

如：*, \=, \+, \{} 等。前面所舉的 .* 的例子，由於 greedy 的關係，在整篇文章中做替換時，會被當成是每一行整行，因為 regexp 會去找每一行最長符合的那一個。

[實例] This is a test. Test for regexp.
如果您下 `:%s/[Tt].*t/program/g` 原意是想把所有的 Test 或 test 換成 program 的，結果由於 regexp 的貪心，整個 "This is a test. Test" 會換成 program。結果原文就變成了 program for regexp. 因此在全文替換時要非常小心，避免使用彈性太大的 regexp。像此例，只要下 `:%s/\<[Tt]est\>/program/g` 就可以了！

最後提醒您，這可不是 regexp 的全部，礙於篇幅及在下功力的問題，當然是沒辦法全面詳盡的向各位做介紹，在下只是將各位領進門，修行就得看各位了！如果還想更深入的研究 regexp，可參考： Mastering Regular Expressions(O'Reilly & Associates) 一書。中文的話，可參考龍門少尉的網站：

http://www.rtfiber.com.tw/~changyj/

裡頭的『正規表示式入門與應用（一、二、三）』，非常值得研讀。

第 11 章

把 *Vim* 折疊（folding）？

把 *Vim* 折疊（folding）後，然後可以放入口袋？呵呵，當然不是這樣啦！這是 *Vim* 的一個很特殊功能（原始 vi 及一般的 vi clone 皆無此功能）。簡單的說，就是可以將文章內容，依據他的結構，把多行內容集中於其中一個代表行來顯示，螢幕上只看得到章節標題那一代表行，這樣整個文章結構就一目了然，真正要閱讀其他內容時，再由簡單的按鍵或滑鼠來打開。這對於不含目錄結構的文件或程式碼很有用處。

11.1 手動折疊

折疊的行為表現是由 `foldmethod[fdm]` 這個設定項來控制的，這個設定項預設是 manual，就是手動設定，這是本節所要敘述的最基本折疊方式。其他折疊方式會在下一節介紹，折疊的方式，其中會有互相衝突的地方，使用上請注意一下。

11.1.1 折疊的產生

手動產生折疊的指令是 `zf`、`zF`、`:fo[ld]`，以下以例子來說明較清楚。

zfap	**將游標所在處的那個段落折疊成一行。**
zf7G	**自游標所在處至全文第 7 行折疊起來。**
:3,9fold	**將第 3 行至第 7 行折疊起來。**
5zF	**將游標所在處起算 5 行的內容折疊起來。**

也可以將所要折疊的部份以 Shift+v 標記起來，然以按 zf 就會將標記的內容折疊起來。

11.1.2 折疊的操作

zo 將游標所在處的折疊打開。**o**pen。
zc 將游標所在處已打開的內容再度折疊起來。**c**lose。
zr 將全文的所有折疊依層次通通打開。**r**educe。
zm 將全文已打開的折疊依層次通通再折疊起來。**m**ore。
zR 作用和 zr 同，但會打開含巢狀折疊（折疊中又還有折疊）的所有折疊。
zM 作用和 zm 同，但對於巢狀折疊亦有作用。
zi 這是個切換，是折疊與不折疊指令間的切換。
zn 打開全文的所有折疊。fold **n**one。
zN 這是 zn 的相對指令，回復所有的折疊。

那這個 zn 和 zR 不是都一樣嗎？不是的，zR 的作用於 foldlevel 這個設定項，是控制折疊的層次，而 zn 則是作用於 foldenable 這個設定項，他是不管層次的，只有全關或全開。當然，所得到的結果有許多情形下是一樣的，但裡子不一樣，這在寫 *Vim* script 時得小心分辨。

通常，游標在折疊處向左或向右移的話，折疊也會自動打開。在進入編輯模式，例如按 i 或 o，這是也會自動打開折疊。

其他的複製、刪除及替換等動作還是可以和一般正常文稿一樣的操作，例如 dd 就會刪掉一整個折疊，yy 會複製一整個折疊，p 會貼上一整個折疊。把整個折疊就當做是一行就行了。

11.1.3 折疊的記憶

手動折疊，在下一次開檔時就消失了，但我們可以使用 `:mkview` 把折疊的情形記憶起來，下一次載入同一個檔案時就可以下 `:loadview`，記憶起以前折疊的情況。當然，使用手動折疊及記憶，在操作上並不是很方便，除非是把他寫成 *Vim* script 來叫用。因此下一節要談到自動折疊，這可能會比較實用一點。

11.2 自動折疊

這裡比較實用的是依文件內的標誌來折疊，因此其他的方式就只稍微介紹了。

11.2.1 以縮行爲依據

`:set foldmethod=indent` **就會依縮行來折疊，本來預設是** `:set foldmethod=normal`**。請注意，此時手動折疊的設定會被停止無法使用。**

11.2.2 以特殊的表示法爲依據

`:set foldmethod=expr`**，另外還要設定** `:set foldexpr=...`**，可參考線上使用手冊** `:h fold-expr` **的例子。**

11.2.3 以語法爲依據

這是在定義語法（syntax**）檔時同時加入折疊的定義。然後，**`set foldmethod=syntax` **就會依照這個語法定義檔去折疊，請** `:h syn-fold`**。**

11.2.4 以是否更改過爲依據

這在進入 `vimdiff`**（即** `vim -d file1 file2`**） 時會自動進入** `foldmethod=diff` **的模式，因此要** `diff` **設定項設在同一個螢幕顯示才行。以便整體的瀏覽。當然要自行更改預設值亦可。預設是：**

```
setlocal diff foldmethod=diff scrollbind nowrap foldlevel=1
```

這樣一來，兩個檔案相同的部份會折疊起來，沒有折疊的部份就是有差異的地方。

11.2.5 以文件上的標誌爲依據

這是跟據文章中的標誌（marker**）來做折疊。這些標誌，起於** {{{**，止於** }}}**，這中間的部份會折疊起來。通常這些標誌是藏在註解欄裡頭，例如** C **程式碼的** /* **及** */ **之間，**shell script **的** # **之後，***Vim* script **的** " **之後，得視程式語言的不同而定，因此一般的文章就不適合使用了。**

這些預設的標誌也可以由 `foldmarker` **來更改，但為了文件的流通性，建議使用預設值就可以了。**

當然，一些程式碼載入時再來設定 `:set foldmethod=marker` 就太麻煩了，這個設定可以設在文件內，例如：

```
#!/bin/sh
# 這是一個 sh script
# {{{
.....
這裡是 script 內容，由 vim 打開這
個檔時，這個部份會自動折疊起來。
.....
# }}}
# vim:fdm=marker:ts=2
```

還記得 modeline 嗎？請複習一下第 9 章，第 9.5 節，頁 38。

第 12 章

Vim tags 的使用

tag 指的是文件中的一種特殊的標誌，在使用 *Vim* 時，可以由很簡單的按鍵就馬上跳到那個文章及那個 tag 的位置，也可以跳回原處。這個功能就好像目前的網頁上的 hyperlink 一樣。但不一樣的是，這個 tag 並不是寫在文件裡頭的，而是由 ctags 這支程式（或其他類此的工具程式）來產生相關檔的 tags，然後存檔於一個外部檔案裡頭，要用到時再由 *Vim* 叫出來。

12.1 各種程式碼專用 tag 工具

ctags	這是最常用到的，可能會有兩種版本，舊的 ctags 只能用於 C 程式碼。exuberant ctags 則可用於 C/C++、Java、Fortran……等等。可由 `ctags --version` 得知版本。
etags	這是 emacs/xemacs 所附的，功能也是非常強大。
JTags	這只能處理 Java 程式碼。
ptags.py	處理 Python 程式碼。
ptags	處理 Perl 程式碼。

這裡主要講述 exuberant ctags。可 `man ctags` 或 `ctags --help` 得知所支援的程式語言。在 `$VIMRUNTIME/tools` 目錄下會有一些工具可以使用，例如專用在 sh script，Tcl/Tk script 的 tag 工具。其他的 tags 工具，系統上不一會安裝，有需要的話得自行安裝，一般使用，應該 `ctags` 就夠用了，`ctags` 也可以模擬 `etags`。

以往，*Vim* 會附上 exuberant ctags，但新近的版本已沒有附上，得由使用者自行安裝，或使用系統上就有的 ctags。如果系統上的並不是 exuberant ctags，可自行由 http://ctags.sourceforge.net 下載、安裝。

12.2 tags 檔案的格式

以下是一般 tag **檔的結構（以一行為例）：**

```
tagname TAB tagfile TAB tagaddress term field
```

tagname	**這是識別字的名稱，通常就是一些函數名，或其他任何識別字。**
TAB	**這是老老實實的一個** Tab **鍵。**
tagfile	**這是** tag **檔的檔名。**
tagaddress	**這是** Ex **指令，通常就是搜尋指令，但行數也是可以。**
term	**;" 這個記號（兩個字元）以後的內容視為註解。**
field	**待瞭解。**

12.3 tag 檔案的製作

不講究的話，可以在所解開的 source code **目錄下，下以下指令：**

```
ctags -R *
```

這樣會有 source code **目錄下產生一個** tags **這個檔（可以使用** -f **選項來指定檔案名），裡頭就包含了整個** source code **的所有檔案的** tags **資訊，包括其下所有的子目錄下的檔案。**ctags **已盡可能的做到聰明掃描檔案的能力，會忽略和程式碼無關的檔案。當然** ctags **還有許多精細的參數可以使用，請** man ctags**。**

請注意，ctags **預設會將輸出檔排序，因此不必自行另外去排序。有排序有一個好處，那就是** *Vim* **會去使用** binary search **的方式去搜尋，這樣會比較快。**

12.4 一般的 tag 使用

如果就照上一節的方式產生 tag files**，那麼只要在** source code **目錄下由** vim **去開啟檔案的話，會自動載入** tags **這個檔案，無需另行載入，否則要由** :set tags=your.tags **來指定** tags **檔。然後就是照一般使用** *Vim* **線上說明一樣，游標移到識別字或函數名上，按** Ctrl+]**，要回到原處就按** Ctrl+T**。**

請注意，*Vim* 啟動時，工作目錄（vim 啟動時的所在目錄）名為 tags 的檔案會自動載入，$VIMRUNTIME/doc 及 $HOME/.vim/doc 目錄下的 tags 檔也會自動載入。而且，凡是載入的 tags 檔裡頭所有標誌文字都可以使用補全鍵來補全，別忘了這個好用的功能。

12.5 *Vim* 線上說明文件的製作

***Vim* 的線上說明文件就是使用 tags 的方式來管理的，因此使用方法也是和一般 tags 檔一樣，由 Ctrl+] 及 Ctrl+T 來控制。**

12.5.1 doctags

這個工具一般不會在系統的 $PATH 裡頭，而是在 *Vim* 原始碼的 runtime/doc 目錄下。由於一般 tags 程式只對程式碼作用，因此對一般的文字檔沒有作用。而這個 doctags 則會依文字檔中有 *這是標題* 標誌的內容做出 tag 檔出來。

在 source code，進入 runtime/doc 目錄後：

```
make doctags => 編譯出 doctags 這個可執行檔
make tags => 製作此目錄下所有 *.txt 的 tags 檔
```

這個 doctags 也可以保留下來，把自己新製作的 *.txt 置於 $VIMRUNTIME/doc 目錄下，執行：

```
doctags *.txt | sort > tags
```

這樣就行了，重新進入 vim 後就可以使用了。

當然，這個工具需自行編譯，因此對一般使用者而言，使用上並不方便。其實 *Vim* 已有內建這個工具了。

12.5.2 由 *Vim* 裡頭作線上說明

進入 vim 後，:helpt[ags] 目錄名 這樣就會把所指定目錄下的所有 *.txt 產生 tags 檔案。這個動作也可以由命令列來執行：

```
vim -c "helptags ." -c quit
```
這和
```
doctags *.txt | sort > tags
```
是一樣的。

由於 *Vim* 自動會搜尋的文件目錄是，目前工作目錄、`$VIMRUNTIME/doc`** 及 **`$HOME/.vim/doc`
因此建議把自己的新文件置於 **`$HOME/.vim/doc` 較好。以下為一個簡單的例子：**

```
This is a test.         *test1*
This is another test.   *test2*
```

存檔成 **`test.txt` 置於 **`$HOME/.vim/doc`** 目錄下：**

```
cd .vim/doc
vim -c "helptags ." -c quit
```

重新開啟 **`vim`，然後 **`:h test1`** 試看看就知道怎麼一回事了。而按 **`F1`** 求助鍵的話，會發現在後面的砲份多了一個章節，那就是 **`LOCAL ADDITIONS:`**，家目錄下的文件目錄就是置放於此。**

第 13 章

Vim script 簡介

原始的 vi **本就有一些簡單的** macro **語法，可以設定一些複雜的編輯動作於一個指令，等於是在創造一個新的指令。** *Vim* **則更進一步把他發展成程式語言。在** *Vim* **中，一般的設定檔，如** vimrc **及一些** syntax/plugin **檔，都是由** *Vim* script **寫成的。可以設定變數，也有迴圈、條件判斷及內建函數可以使用，更可以自訂函數，儼然就是一個程式語言雛形了。當然，原始的** vi macro **語法還是認得的，這點不必擔心。**

由於 *Vim* script **的大大擴充，雖然說是簡介，但內容可能會比其他的章節多，而且可能會比較深入一些，可以視情形，撿幾個順手的例子來用，不必一開始就要通通搞懂他。**:-)

13.1 一些簡單的 macro

這裡指的 macro **是一般的鍵盤對應、縮寫設定及簡單的新命令定義，雖然也是會使用到** *Vim* script**，但由於並不是完整的規劃、設計，因此就以** macro **為名來代表（當然，實質上也是利用** *Vim* script **來書寫的）。**

13.1.1 按鍵對應

Vim **可以將多個動作（命令或是函數）對應給一個簡單的按鍵，這樣一來就可以很方便的按個鍵去執行所定義的動作，基本的例子就是第** 9.7 **節，頁** 39 **的一些實際例子。由於** *Vim* **預設已經內建把** F1 **鍵定義給線上求助檔，因此，這個鍵就不要去麻煩他了。**

一些按鍵的書寫方法

在傳統的按鍵對應，有他的簡單書寫方法，但通常並不是很直覺，*Vim* 則改進了書寫方法，可以很直覺的書寫。以下是一些例子：

<**Esc**>	Esc **鍵。**
<**Tab**>	Tab **鍵。**
<**Home**>	Home **鍵。**
<**Del**>	Del **鍵。**
<**CR**>	Enter **鍵。**
<**Enter**>	Enter **鍵，和上面的相同。**
<**LT**>	**就是** <，**在和** < **字元本身會有混淆、疑義時使用。也可以使用** \<。
<**BS**>	Backspace **倒退鍵。**
<**Up**>	PageUp **向上翻頁鍵。**
<**F5**>	F5 **功能鍵。**
<**C-G**>	Ctrl+G **鍵。**

要注意的是，大小寫是不分的，以上的書寫只是為了閱讀方便而已。詳細的按鍵符號，這裡就不多說明了，有需要的話，可以進入 `vim` **後** `:h key-notation`，**就會有詳細的列表。**

\< **的使用只能在** `cpoptions` **設定項不含** B **旗標時才能使用，通常** B **旗標預設是會有的，請** `:set cpoptiions?` **就可知道目前的設定。**

要非常小心的是，這種直覺的書寫方式，不能用於 :set **及** :autocmd **的情形，因為角括號在裡頭有其他的特殊意義。通常，這只能用於按鍵的對應，縮寫及選單設定的情況。**

map 指令的種類

Vim **除了原始的** `map[!]` **外，擴充了相當多的類似** map **指令。各在不同的模式作用。**`map` **主要是作用於常態模式及反白模式，而** `map!` **則是作用於插入模式及命令列模式。**

vmap	**僅在反白模式時作用，**Visual mode**。**
nmap	**僅在常態模式時作用，**Normal mode**。**
imap	**僅在插入模式時作用，**Insert mode**。**
omap	**僅在操作等待模式時作用，**Operator-pending mode**。**
cmap	**僅在命令列模式時作用，**Command-line mode**。**

[**實例**] `:map <F5> bi{<Esc>ea}<Esc>`
這樣一來，只要在英文單字任意處按 F5**，這個英文單字的前後就會加上大括號。**

說明如下：

bi{<Esc>	按 b 移至英文單字的第一個字母處，並按 i 進入插入模式，寫入 { 這個字元，並按 Esc 鍵，回復常態模式。
e	移動至這個英文單字的最後一個字母處。
a}<Esc>	進入插入 append 模式，並輸入 } 這個字元，再回到常態模式。

需注意的是，在命令列模式前頭會有個冒號，因為在 `vim` 裡頭，要按個冒號才會進入命令列模式。但如果是設在 *Vim* script 檔裡頭當然就沒有冒號了。以上的例子也是可以對應至一些較容易記憶的按鍵，例如：

```
map ,b bi{<Esc>ea}<Esc>
```

這樣按 ",b" 就會有同樣的作用，但 "," 及 "b" 之間不能相隔太久的時間，這樣 *Vim* 才會知道這兩個按鍵是結合在一起的[1]。而 b 在此就代表 braces（大括號）。

如果要知道目前的按鍵對應的情形，可 `:map` 就會列出所有的目前按鍵對應，前面標示的記號就是各種模式的代表字母，例如 n 代表 normal，i 代表 insert。

防止重複對應

為了防止重複對應到已有定義的按鍵，通常指令內可加入 `nore` 的字樣，例如：

```
:noremap  => Normal, Visual and Operator-pending
:vnoremap => Visual
:nnoremap => Normal
:onoremap => Operator-pending
:noremap! => Insert and Command-line
:inoremap => Insert
:cnoremap => Command-line
```

其實其他對應性質的指令也是會有類似的這種指令，請參考下一節。

13.1.2 縮寫對應

縮寫對應是把一長串的字串對應到簡單的幾個代表性字串。縮寫的對應只能用在插入模式、取代模式及命令列模式。主要用於輸入時節省時間及避免拼錯。這裡利用例子來做說明：

[1] 可以在底下的狀態列看得到按鍵，按 "," 會顯示出來，等一段時間後又會消失，在這段時間內按 "b"，那 *Vim* 就會把他當做兩個鍵是結合在一起的。

[實例] `:ab gl GNU Linux`
把 gl 這個按鍵對應成 GNU Linux。

ab 是 abbreviate 的縮寫，這樣一來，在插入模式時，只要輸入 gl，再按 Sapce 鍵或 Ctril+]，這個 gl 就會變成 GNU Linux 這個字串。Sapce 和 Ctril+] 的區別在於前者會多留個空白，方便繼續輸入其他文字；而後者則不會多留個空白。 要注意的是，這在取代模式（請參考第 3.1.2 節，頁 11）、命令列模式（請參考第 1.4 節，頁 4）也是可以作用，如果是使用 `:ia[bbrev]`，那只會在插入模式及取代模式有作用。

以下是一些常會用到的指令：

`:ab[breviate]`	不接任何參數，這會列出目前所有的縮寫對應。其中標有 'i' 的，代表作用於插入模式；標有 'c' 的，代表作用於命令列模式；標有 '!' 的，則兩種模式皆有作用。
`:ab gl`	這會列出 gl 是對應成什麼字串。
`:una[bbreviate] gl`	取消 gl 這個對應。
`:ia[bbrev]`	和 `:ab` 的定義一樣，但只作用於插入模式。
`:cb[bbrev]`	和 `:ab` 的定義一樣，但只作用於命令列模式。
`:norea[bbrev]`	用於防止重對應已有對應的字串。
`:abc[lear]`	取消所有由 `:ab` 所定義的對應。
`:iabc[lear]`	取消所有由 `:ia` 所定義的對應。
`:cabc[lear]`	取消所有由 `:ca` 所定義的對應。

13.1.3 定義新命令

Vim 提供自行定義新命令的方法，其語法是：

`:com[mand][!]` [屬性……] 新命令名 動作

屬性的部份較複雜，在此先省略不談，底下再來詳細說明。其中 '!' 代表強制定義已有的命令，否則已存在的命令是不允許重定義的。新命令名也是有限制：

(1) 必需以大寫英文字母開頭。
(2) 不能使用 `:X`，`:Next`，`:Print`，這三個保留字。
(3) 其他的部份，可以是字母或數字，分大小寫，但不能使用底線。
但不建議使用數目字。

通常此類新命令的定義是使用在 *Vim* script 的場合。

13.1.4 新命令的屬性

Vim 會把這些命令當成是 ex 命令一般的地位。這裡的所謂屬性指的是新命令所能使用的參數及是否可以指定範圍等等。這裡分成四個部份來說明：參數、補全動作、範圍指定及特殊情形。

參數

新命令可以使用 -nargs 屬性來指定參數的性質。

```
-nargs=0  這是預設，就是沒有參數，可以省略不寫。
-nargs=1  需要一個參數。
-nargs=*  是否有參數及參數數目不拘。
-nargs=?  允許零或一個參數。
-nargs=+  必須有參數，但參數數目不拘。
```

參數之間以空白或 Tab 區隔。實際上定義時，置放參數的位置是以 <args> 來代表的。

```
<args>   一般使用，但參數內無法使用雙引號。
<q-args> 和 <args> 同，但可以使用雙引號。
<f-args> 用於函數內所要用到的參數。
```

13.2 *Vim* script 的語法

Vim script 只有兩種資料型態，數字及字串。

第 14 章

Vim 和其他軟體的配合

編輯器畢竟是有其特定功能，無法包山包海的把一些電腦上會用到的功能通通整合進去。但由於 *Vim* 提供了 *Vim* script 及和外部 shell 溝通的功能，因此很容易就可以把 *Vim* 和其他軟體結合起來使用，不必在編輯文件時還要退出 *Vim* 去執行其他應用軟體。

14.1 和郵件、新聞軟體的配合

這裡只以個人使用的 mutt/slrn 為例，其他的類似軟體就請各位發揮想像力囉！

14.1.1 mutt + vim

```
# ~/.muttrc
set editor="vi -c ':0;/^$'"
```

14.1.2 slrn + vim

```
# ~/.slrnrc
set editor_command "vi -c ':0;/^$' '%s'"
```

這個 `-c` 參數的意思，就是進入 *Vim* 後，馬上執行參數後的 *Vim* ex 命令。在此是 `:0;/^$`，什麼意思？該去複習一下規則表示式囉！這會移至信件表頭與信件實際內容的中間（就是全文中第一個遇到的空白行），以便可以方便馬上就地編輯。首先 `:0` 是移到文件的第一行，然後 `/^$` 是找第一個空白行。

14.2 和編譯程式的配合

這和一般使用者的關係不大，但對於寫程式的人來說就非常方便了。

14.3 和 TEX/LATEX 的配合

14.4 和 Java 的配合

第 15 章

Vim tips 集錦

本章是一些實際上常用、好用的 tips **集錦。當然每個人的使用習慣並不一樣，因此這些** tips **不可能適合每一個人，但可以經由修改，改成自己滿意的使用方法，這才是本章的重點所在。**

授權聲明

參考書目

[1] Bram Moolenaar, “VIM USER MANUAL”

[2] Edward G.J. Lee, “Learnning Vim”

[3] Steve Oualline, “Vi IMproved - Vim”

索引