大家来学 VIM（一个历久弥新的编辑器） [一]

前言

鉴于仍有许多人还没找到顺手的编辑器，而许多想学 vi 的人又觉得无从下手，因此在此提出一些个人的心得，希望能对这些朋友有点帮助。或许也减少一点 FAQ 吧！

真要深入的话，大多数的前辈都认为 vi 比 emacs(xemcas) 还难学。但谁又真的需要熟悉编辑器的所有功能呢？你大可以边用边学啊！需要用到的先学，其它的就放一边，只要能善用一些常用到的功能，又何必要那么深入呢？而且您在使用当中经常会发现一些新功能，这又会马上让您给赚到了。

为什么选 VIM ？

最最重要的原因是可以正确处理中文！其它如 elvis,vile,nvi 在中文方面都会有问题。另外就是许多操作系统都有 VIM 可用。当然如果您不需要中文支持的话，也建议使用 elvis。vile 则有 emacs 的味道，而 nvi 大概是最忠于原味的了！至于原始 vi 的书已有中文翻译（O'Reilly），各位到大一点的书局翻翻就有了。所以选定 VIM 做对象，兼述及 elvis，至少她不「排斥」中文。

- VIM 代表 Vi IMproved。
- elvis 可直接读 HTML(可用来上网，但只有文字)，binary，manpage 及 TeX(LaTeX)(不是只显示程序代码喔！)檔，和 XA+CV 配合也可以用中文，但不能真正处理就是了！
- VIM 也不是真的能完全支持中文啦！有些细部的功能还是没法度。大概 6.0 版的时候会更好！不过平常一般的编辑动作是没问题。

另一个原因是，VIM 是慈善软件（CharityWare），如有赞助或评比得奖（前不久刚得过），所得皆救助乌干达孤儿，有点年纪的大概还记得"We Are The World" 这首歌吧！全球的艺人共同合唱的，就是救助这些孤儿（应该没记错吧！）。您要使用当然是免费，您要捐款赞助当然是不勉强啦！但如果有评比有奖金可拿的，您去投她一票就是功德无量了。

另外 vim 的规则表示式(regular express)颇完整，您也可借这个机会学 regexp，因为您在 sed,awk,perl,less,grep...中也是要用到，早点会 regexp 您生活在 Linux(Unix)的世界会更美好。学了 regexp 您会有点看不起 windows 系统中的寻找功能的。

由于是慈善软件，广结善缘，因此连中文繁体都支持，不简单。但也因此最近的版本有点肥，但又不会太肥(比 xemacs 好多了啦！)。这么可爱的软件，能不用她吗？

何处抓 VIM(elvis)？

http://www.vim.org/

找个自己中意的 mirro 站抓。或许也顺便抓 Win32 的可执行

文件回来在 windows 系统中使用。别忘了 runtime 檔也要抓，

否则会无法找到需要的档案来执行。

http://www.fh-wedel.de/elvis/index.html

ftp://ftp.pdx.edu/pub/elvis

也可试试 elvis（当然是免费的）。

CLE 的使用者也可以到：

ftp://linux.tmtc.edu.tw/pub/CLE/i386/RedHat/RPMS

抓最新版来安装，省时省力，而且还把必要的中文设定都弄好。

目前最新的版本，VIM 是 5.6，elvis 是 2.1-4。

编译安装

只讲重点，避免啰嗦！

VIM：

1. configure 时加上：

   --enable-multibyte

   --enable-xim　如果您使用 xcin2.3 的话，就不必了

   --enable-fontset

   --enable-gui=gtk

   gtk+ 最好是 1.2.3 以上的版本，1.2.1 也是将就可以啦

   ！大家都用 gtk+ 了，您不会想编 Athena 或

Motif(lesstif)吧！而且后两个版本的 GUI 就没有漂亮的

icon 了（但有的人就是讨厌这些无用的 icon）。

2. ~/.vimrc 加入：

set fileencoding=taiwan

set guifontset=英文字型，中文字型

设了 guifontset 就不能设 guifont 否则会优先使用

guifont，这样就找不到中文字型了！console 下或 xterm

下的话，就看您用什么字型就显示什么字型，和

guifontset 无关。

- 英文字型要用固定字，建议使用危卵的 Andale Mono 这个字型，包您满意，可去危卵抓，免费的。我个人是使用 180 的 Andale 字，配上 220 的中文字，绝配(1024x768)！哦！不要告诉我，您的 X server 还不能支援 TTF。
- Windows 中文版无需设 guifontset，只要设 guifont 即可。

3. 使用 xcin2.3 的话还是得配合 XA+CV 主要是输入的问题。

使用 xcin2.5 的话就不必 XA+CV 了，但您得另外设 xim，

请进入 VIM 后 :help xim。

4. locale 要设成 zh_TW.Big5，意思就是您的系统的 i18n

支持不能太差，我是使用 gvim-chinese shell script

来呼叫 gvim。

```
#!/bin/sh
# gvim-chinese
# unset LD_PRELOAD       如果您使用 xcin2.5x 的话
```

```
export LC_ALL=zh_TW.Big5

gvim "$@"
```

elvis:

elvis 没什么好说的，反正是靠 XA+CV 来显示及输入中文

。至于其它外观调整，就请 man elvis。

勤前 教育

VIM 的模式可有六种，为免初学的人搞混，仍依一般的分类分成三种：

1. Normal mode(common mode，以下简称 c-mode)：

   您一进入 VIM 就是处于 c-mode，只能下按键指令，不能输入文字。这些指令可能是光标移动的指令，也可能是编辑指令或寻找替换指令。

2. Insert mode(i-mode)：

   按 i 就会进入 i-mode，此时才可以键入文字，写您的文章，按 Esc 又会回到 c-mode。

3. Ed mode(common-line mode，e-mode)：

   按冒号：(别忘了 Shift 键)就会进入 e-mode，左下角会有一个冒号：出现可下 ed 指令。也是按 Esc 回 c-mode。反正正常状态都是处于 c-mode，这样才不会把您辛苦打字的文章给随便搞乱掉。

   - ed 是一个很古老的行编辑器，就好像 DOS 下的 edline 一样，vi,sed 这些编辑器都是从 ed 衍化而来的。当然 DOS 下的 edline 也是学 ed 而来的，但功能可就不能同日而语了！有兴趣可 man ed 学看看，有些时候会只有 ed 可以用（当机救助的时候）。

其它的模式就碰到时再说明了！

基本教练 ： step by step

1. 由命令列来开档：

   vim test.txt 或 gvim test,txt 或 gvim-chinese test.txt 如果您的系统 vi 是 vim 的连结档话，就可以直接用 vi test.txt。以下就直接用 vim 代表 vi，gvim，elvis 因操作是一样的有不同的地方会加注说明。

2. 先进入 vim 后再开档：

进入 vim 后，使用冒号命令 :e test.txt，就可以编辑 test.txt 这个档。1. 及 2. 这两个开档法，如果 test.txt 不存在的话，就会开一个以 test.txt 为名的新档案。

- 如果是 gvim，可由 icon(gtk 版本才有)或菜单来叫出 file browser 来选看看您要编辑哪一个档，但如果您是初学的话不建议您这么做，vi 就是以按键快速闻名，这是她的优点，您还是学起来吧，不然没有 GUI 的时候会很不习惯的。

3. 编写文件：

进入 vim 后，按 i 进入 i-mode，就可以编写您的文件了。在 vim 光标的移动可以由方向键来移动。Backspace 键可消去前一个字符，中文的话是一个中文字。Del 键可删除光标所在处的字符（中文字）。

- 原始 vi 是不能在 i-mode 随意移动光标的，得进入 c-mode 才能移动，因此就常常要按 Esc 来变换模式。vim 及 elvis 都打破了这个规矩。

4. 存档、离开：

如果您写好您的文件，就可以按 Esc 回到 c-mode，然后 :w 就会存盘（注意，是冒号命令），但还不会离开 vim，要离开可按 :q，就可以了！也可以合起来用，:wq，就样就会存档后离开。怎么样，也不会很难吧！只不过操作方式和别的编辑器不一样罢了，这样岂不是很有个性。:-)

- 尽量记住按键的意义，才不必死背，如 e 是 edit(编辑), w 是 write(写入)，q 是 quit(停止、离开)。

好了，这是就编辑的整个过程。下回开始是详述各部份的功能，把 vim 解剖开来讲，您可以马上现学现卖。由于 vi(m) 的操作方式很有个性，因此，用了一次就会记住有这么一个功能，想忘也忘不了，但有时按键难免会忘记，但有这一种功能大概是忘不了的，查了几次指令就可以记得住了。

大家来学 VIM（一个历久弥新的编辑器） [二]

光标的移动

本节所述皆是在 common-mode(c-mode，在 vim 又名 normal-mode)下的移动，原始的 vi 只能在 c-mode 移动光标，在 insert-mode 只做文字的输入，而不做光标的移动。当然 vim 及 elvis 的方向键是不论在那一种 mode 皆可移动自如。

基本的光标移动

h 左，或 Backspace 或方向键

j 下，或 Enter 或 +（要 Shift 键），或方向键

k 上，或 方向键或 –（不必 Shift 键）

l 右，或 Space 或方向键

- 使用 hjkl 键的移动是为了使手不必离开打字区（键盘中央的部位），以加快打字的速度，如果各位不习惯，那就使用方向键吧！
- Backspace 及 Space 的移动方式是到了行首或行尾时会折行，但方向键或 hl 键的移动则在行首或行尾时您继续按也不会折行。转折换行的功能是 vim 的扩充功能，elvis 无此功能。
- jk 及使用方向键的上下移动光标会尽量保持在同一字段。使用 Enter，+，– 的上下移动，光标会移至上（下）一行的第一个非空格符处。
- 好像有点复杂，各位就暂时使用方向键来移动就简单明白了！等您爱上了 vim 后再来讲究吧。

Ctrl–f 即 PageDown 翻页。

Crtl–b 即 PageUp 翻页。

进阶的光标移动

0 是数目字 0 而不是英文字母 o。或是 Hmoe 键，移至行首，

（含空格符）。

^ 移至第一个非空格符，注意，要 Shift 键。

$ 移至行尾，或 End 键。要 Shift 键。

- 以上两个按键是源自规则表示式（regular expression），在 regexp 中 ^ 是匹配行首，$ 是匹配行尾。

G 移至档尾（最后一行的第一个非空格符处）

gg 移至档首（第一行之第一个非空格符处）

- gg 是 vim 的扩充功能，在 elvis 或原始 vi 中可用 1G 来移至文件首（是数字 1 不是英文字 l）。
- G 之原意是 goto，指移至指定数目行之行首，如不指定数目，则预设是最后一行。

w 移至次一个字（word）前缀。当然是指英文单字。

W 同上，但会忽略一些标点符号。

e 移至前一个字字尾。

E 同上，但会忽略一些标点符号。

b 移至前一个字前缀。

B 同上，但会忽略一些标点符号。

H 移至屏幕顶第一个非空格符。

M 移至屏幕中间第一个非空格符。

L 移至屏幕底第一个非空格符。

- 这和 PageDown，PageUp 不一样，内文内容并未动，只是光标在动而已。

nl 移至第 n 个字符(栏)处。注意，要用 Shift 键。n 是从头起

算的。

:n 移至第 n 行行首。或 nG。

特殊的移动

) 移至下一个句子（sentence）首。

(移至上一个句子（sentence）首。

} 移至下一个段落（paragraph）首。

{ 移至上一个段落（paragraph）首。

- sentence 是以 . ! ? 为区格。
- paragraph 是以空白行为区格。

% 这是匹配 {}，[]，() 用的，例如您的光标现在在 { 上

只要按 %，就会跑到相匹配的 } 上。写程序时满好用的。

另还有一些 vim 的特殊按键，但这得留待最后才来讲述，否则各位恐怕会头昏眼花了。

大家来学 VIM（一个历久弥新的编辑器） [三]

基本编辑指令

这个单元就开始进入主题了。下编辑指令都是在 commond-mode(c-mode)，就是您一进入 vim 时的模式，只能下指令，不能键入文字。如果印象模糊，请瞄一下第一个单元的内容。这个单元说的是基本的指令，有些比较特殊的编辑指令，因为太有个性了，所以会独立成一个单元来说明。

进入 i-mode 的指令

i 在光标所在字符前开始输入文字(insert)。

a 在光标所在字符后开始输入文字(append)。

o 在光标所在行下开一新行来输入文字(open)。

I 在行首开始输入文字。

- 此之行首指第一个非空格符处，要从真正的第一个字符处开始输人文字，可使用 0i 或 gI(vim)。

A 在行尾开始输入文字。

- 这个好用，您不必管光标在此行的什么地方，只要按 A 就会在行尾等着您输入文字。

O 在光标所在行上开一新行来输入文字。

J 将下一行整行接至本行(Joint)。

- 并无相对的 split 功能，可在 i-mode 下按 Enter 来达成，当然如果您熟 macro 的话，可自行定义。
- 使用 J 时，预设会消去本行的 EOL，且上下行接缝间会留下一个空格符，这符合英文习惯，却对中文会造成困扰，欲不留空格符，可使用 gJ（大写 J） 指令，但这是 vim 的扩充功能，elvis 不适用。
- 请您随便找一个档案来试看看，光看文字说明太抽象了。

删除指令

x 删除光标所在处之字符。在 vim 及 elvis 亦可用 Del 键。

X 删除光标前之字符。不可使用 Backspace 键。

- vim 可以正确使用以上两个指令于中文，会删去一个中文字。elvis 则不行，一个中文字要删两次，即使用 xx。

dd 删除一整行(delete line)。

dw 删除一个字(delete word)。不能适用于中文。

dG 删至档尾。

d1G 删至档首。或 dgg(只能用于 vim)。

D 删至行尾，或 d$（含光标所在处字符）。

d0 删至行首，或 d^（不含光标所在处字符）。

- 请回忆一下 $ 及 ^ 所代表的意义，您就可以理解 d$ 及 d^ 的动作，这就是 vi(m) 可爱之处。

取代及还原

r 取代光标所在处之字符。vi(m) 很有个性的，您在 c-mode 按

了 r 她就会停在那里等主人键入所要替代的字符，希望您这

个当主人的，不要傻呼呼的也楞在那里，赶快键入您的新字符

吧！:-)

- vim 中可用于中文字，也就是可以替换一个中文字，elvis 则不行。当然您的 vim 是要设在 taiwan 的才行。怎么样！有没有看过如此有个性的取代方式？丫！r 就是 replace 啦！

R 取代字符至按 Esc 为止。

cc 取代整行内容。或大写 S 亦可。

cw 替换一个英文字(word)，中文不适用。(change)

~ 游标所在处之大小写互换。当然不能用于中文。别忘了 Shift！

C 取代至行尾，即光标所在处以后的字都会被替换。或 c$。

c0 取代至行首，或 c^。

s 替换一个字符为您所输入的字符串。和 R 不同，R 是覆盖式的取

代，s 则是插入式的取代，您可亲自实验看看。ㄟ！是小写的

s。

u 这个太重要了，就是 undo，传统的 vi 仅支持一次 undo，vim

及 elvis 就不只了，vim 是没有限制的。

U 在光标没离开本行之前，回复所有编辑动作。

Crtl+r 这个也是很重要，就是 redo 键。

加上数目字

喔！骚到 vi(m) 的痒处了，这是 vi(m) 一个非常骚包的功能，只此一家别无分号（当然同源的 ed，sed 等不在此限）。就是您可以在大部份的指令前加上数目字，代表要处理几次的意思。以下用实例来说明比较清楚。

5dd 删除游标所在处（含）起算以下五行内容。妙吧！

3r 按了 3r 后，您键入一个英文字，则三个字符皆会被您所键

入的英文取代。很抱歉，这不能用于中文。

5J 将五行合并成一行。

3x 删除三个字符。抱歉，不能用于中文。

5i A 然后按 Ecs，插入五个 A。中文也可以！

2i system Esc 插入 systemsystem。中文也可以！

5G 光标移至第五行，是从档首开始起算。

5l 移至右第五个字符处，当然 j 是可以用方向键取代的。

所有移动指令（参考第二单元）都可以加上数目字来控制，中

文也通喔！elvis 当然是不能用于中文。

其它的指令和数目字结合，就留待各位去发掘吧！最重要的是请您亲自操作看看，使用 vi(m) 常常要动动脑筋，会有更妙的操作方式。

简单 重排功能

>> 整行向右移一个 shiftwidth（预设是 8 个字符，可重设）。

<< 整行向左移一个 shiftwidth（预设是 8 个字符，可重设）。

- :set shiftwidth? 可得知目前的设定值。:set shiftwidth=4 可马上重设为 4 个字符。shiftwidth 可简写成 sw。ㄟ，别忘了 Shift 键！

:ce(nter) 本行文字置中。注意是冒号命令！

:ri(ght) 本行文字靠右。

:le(ft) 本行文字靠左。

- 所谓置中、靠左右，是参考 textwidth(tw) 的设定。如果 tw 没有设定，预设是 80，就是以 80 个字符为总宽度为标准来置放。当然您也可以如 sw 一样马上重设。

gqip 整段重排。中文会出槌！:-(

gqq 本行重排。

- 重排的依据也是 textwidth。这里的重排是指您键入文字时没有按 Enter 键，就一直在 keyin，这样会形成一个很长的一行（虽然屏幕上会替您做假性折行），重排后，则会在每一行最后加入 EOL。gq 重排功能是 vim 才有的功能。

大家来学 VIM（一个历久弥新的编辑器） [四]

复制（ yank ）

yank 是什么意思？有疑问的请查一下字典吧！就好像是中医治疗中的「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 vi(m) 中，她就是复制 copy 的意思。这在 vi(m) 的思考逻辑里，就是「拔」yank 起来，「放」put 上去。其实复制的指令就是 y 一个而已，为什么要独立成一个单元来说明呢？因为 vi(m) 复制、贴上的功能实在太独特了，再配合第三单元介绍的数目字，及 vi(m) 内部的缓冲区来使用的话，您会发现，原来 vi(m) 肚子里还暗藏着秘密武器。

指令说明

yy 复制游标所在行整行。或大写一个 Y。

2yy 或 y2y 复制两行。ㄟ，请举一反三好不好！:-)

y^ 复制至行首，或 y0。不含光标所在处字符。

y$ 复制至行尾。含光标所在处字符。

yw 复制一个 word。

y2w 复制两个字。

yG 复制至档尾。

y1G 复制至档首。

p 小写 p 代表贴至光标后（下）。

P 大写 P 代表贴至光标前（上）。

- 整行的复制，按 p 或 P 时是插入式的贴在下（上）一行。非整行的复制则是贴在游标所在处之后（前）。

"ayy 将本行文字复制到 a 缓冲区

- a 可为 26 个英文字母中的一个，如果是小写的话，原先的内容会被清掉，如果是大写的话是 append 的作用，会把内容附加到原先内容之后。
- " 是 Enter 键隔壁的那一个同上符号（ditto marks）。

"ap 将 a 缓冲区的内容贴上。

- 缓冲区的术语在 vim 称为 registers，vim 扩充了相当多的功能，有兴趣深入的朋友请 :h registers。您用

d、c、s、x、y 等指令改变或删除的内容都是放在 registers 中的。例如：您用 dd 删除的一行，也是可以使用 p 来贴上的。只要是在缓冲区的内容都可以使用 p 来贴上，不是一定要 y 起来的内容才能用 p。因此您认为 p 是 paste 也可以，认为是 put 可能较正确。

5"ayy 复制五行内容至 a 缓冲区。

5"Ayy 再复制五行附在 a 内容之后，现在 a 中有十行内容了！

- 乁！不要我一直用 a 您就认为只有 a 可以用喔。26 个英文字母都可以的，交叉运用下，您会发觉 vi(m) 肚量不小。
- 问题来了！忘记谁是谁的时候怎么办？ :reg（冒号命令）就会列出所有 registers 的代号及内容。您现在就试着按看看。咦！怎么还有数目字、特殊符号的缓冲区，原来您刚刚删除（复制）的内容就预设放在 " 这个缓冲区，然后依序是 0,1,2,...9。也就是说您按 p 不加什么的话，是取出 " 缓冲区的内容的。% 指的是目前编辑的档案，# 指的是前一次编辑的档案。还有其它的呀！因为没什么重要，就请 :h registers 吧！registers 有个 "s" 结尾，不要搞错了，而且 Tab 的补全键 vim 也支持的，也就是说您键入 :h regi 再按 Tab 键，vim 就会帮您补全，按了 Tab 后发现不是您要的，那就继续按，总会出现您要的。:-)
- Tab 补全的功能，elvis 也有，但叫出 registers 列表的命令则没有，您得自行记忆在您的脑袋瓜子里。而且 elvis 的补全能力并没 vim 强。

天大的指令

. 这是什么？丫，是英文句点啦！没错，就是英文句点。什么意思？重复前次的编辑动作。这个指令太高明了，只要是编辑动作（移动光标不算，冒号命令也不算）都可以按英文句点来重复，要重复几次都可以。

例如：您按了 yy，然后按 p 就会复制、贴上一整行，如果要重复这个动作的话，就可以按 . ，也可以把光标移到其它地方后再按。其它 dd，dw，r，cw 等编辑指令都可以这样来重复。如果您要重复做某些编辑动作时，千万千万一定要想到有这么一个英文句点重复指令。丫，拜托啦！您一定要常用这个指令。

疑难杂症

1. 那 mouse 中键的剪贴功能还有吗？

当然还有，不管在 console 或 X terminal 中都照用不误。当然在 windows 下的话就不

能用了，可以用 Shift-Insert 来代替。Ctrl-v 在 vim 中另有作用，在 windows 下就不必去麻烦它了。

2. 飞，我从 netscape 用 mouse copy 东东过来的时候，常常都搞得天下大乱耶！

   要设成 :set paste，预设是 map 至 F9 键的，您要 copy 之前先按一下 F9，copy 完后再按一次 F9 来回复。这是 vim 的扩充功能，elvis 没有。那在 elvis 怎么办？只好 :set noai 了。在 GUI 的版本应不会有这种情形。

   - set 的功能先不必去理它，往后会有一个单元专门讨论。

朋友！您睡着了吗？不要被吓到了，您只要开个档案，亲自操作一下就能心领神会。那用 mouse 不是更方便吗？不见得，yyp 来复制贴上一整行比较快，还是用 mouse 来拉比较快？您可以试看看。

大家来学 VIM（一个历久弥新的编辑器） [五]

寻找、替换

搜寻、替换的功能几乎是每个编辑器必备的功能，那在 vi(m) 中有没有特殊的地方呢？当然有，您忘了，vi(m) 是个性十足的编辑器。最特殊的地方是和规则表示式（regular expression,简称 regexp）结合在一起。简单的说她是一种 pattern 的表示法，在执行动作，如寻找或替换，就会依据这个 pattern 去找，所有符合 pattern 的地方就会执行您所下的动作。在这个单元里暂不讨论 regexp，会另立一个单元来探讨，以免搞得头昏脑胀。目前就暂不使用 regexp，您要找什么就直接键入什么就对了。

寻找

/ 在 c-mode 的情形下，按 / 就会在左下角出现一个 /，然后键

入您要寻找的字符串，按个 Enter 就会开始找。

? 和 / 相同，只是 / 是向前（下）找，? 则是向后（上）找。

n 继续寻找。

N 继续寻找（反向）。

更方便的寻找 操作（ vim 才有）

* 寻找游标所在处之 word（要完全符合）。

同上，但 * 是向前（下）找，# 则是向后（上）找。

g* 同 * ，但部份符合即可。

g# 同 # ，但部份符合即可。

- n, N 之继续寻找键仍适用。

替换（ substit ute ）

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

range 指的是范围，1,7 指从第一行至第七行，1,$ 指从第一行

至最后一行，也就是整篇文章，也可以 % 代表。

- 还记得吗？ % 是目前编辑的文章，# 是前一次编辑的文章。

pattern 就是要被替换掉的字符串，可以用 regexp 来表示。

string 将 pattern 由 string 所取代。

c confirm，每次替换前会询问。

e 不显示 error。

g globe，不询问，整行替换。

i ignore 不分大小写。

- g 大概都是要加的，否则只会替换每一行的第一个符合字符串。可以合起来用，如 cgi，表示不分大小写，整行替换，替换前要询问是否替换。

[实例] :%s/Edwin/Edward/g

这样整篇文章的 Edwin 就会替换成 Edward。

更进阶的寻找、替换的例子在说明 regexp 的时候还会再详述。目前只知道最基本的用法就可以了！其实光这样就非常好用了。:-)

书签 功能

这又是 vi(m) 的一个秘密武器，简单的说，您可以在文章中的某处做个记号（marks），然后跑到其它地方去编辑，在呼叫这个 mark 时又会回到原处。妙吧！

mx x 代表 26 个小写英文字母，这样光标所在处就会被 mark。

`x 回到书签原设定位置。

- ` 是 backward quote，就是 Tab 键上面那一个。

'x 回到书签设定行行首。

- ' 是 forward quote，是 Enter 键隔壁那一个。

vim 对于书签的扩充功能

小写字母 只作用于单一档案内。

大写字母 可作用于编辑中之各档案间。

数目字 可作用于前次编辑的十个档案。

- 数目字的用法比较特殊，'0 是回到前一次编辑档案中离开前的最后位置，'1 则是回到前二次编辑档案的最后位置，依此类推。您不必使用 m 来标示，vim 会自动记忆。很玄吧！其实这是 viminfo 的功能，您要认真追究的话，请 :h viminfo-file-marks。viminfo 关掉，就没这个功能了！
- 所谓前次指的是前次启动的 vim。

:marks 得知目前所有书签的列表。

大家来学 VIM（一个历久弥新的编辑器） [六]

叫档、存档、紧急回复

飞，是不是在灌水呀！怎么开个文件也成一个单元？那您就错了，在 vi(m) 里叫档的花样可多了，而且又可以多档编辑，各编辑中的档案还可以互通讯息，这里面学问可大着呢！vim 就更骚包了，也学人家档案可以加密，虽说是噱头，但也还满好用的。

开档的一些花招

vim + 文件名 这样开文件后，游标会落在档案最后一行的行尾，在档

案屁屁后干什么呢？方便您可以继续编辑嘛！:-)

vim +n 文件名 游标会落在第 n 行的行首。

vim +/string 檔名

- 还记得吗？ / 就是寻找指令，这样进入档案后光标就会落在第一个找到的 string 上，还可以按 n 继续找 string 喔！哦，string 还可以使用 regexp 来表示喔。

多档编辑

多档编辑会有两种情形，一种是在进入 vim 前所用的参数就是多个档（这种情形称为 argument list）。另一种情形是进入 vim 后另外再开其它的档（称为 buffer list）。不过都可以统称为 buffer。

:n 编辑下一个档案。

:2n 编辑下二个档案。

:N 编辑前一个档案。

- 注意，这种用法只能用于 argument list 的情形。

:e 檔名 这是在进入 vim 后，在不离开 vim 的情形下再开其它档

案。只要您要编辑的档案是在目前目录，Tab 补全键还是

可以使用。

:e# 或 Ctrl–^ 编辑前一个档案，用于两档互相编辑时相当好用。

- 这种用法不管是 argument list 或 buffer list 档案间皆可使用。
- 还记得吗？ # 代表的是前一次编辑的档案。

:files 或 :buffers 或 :ls 会列出目前 buffer 中的所有档案。

- 在 elvis 中可使用 :b 来叫出 buffers。
- 在 buffers 中，减号 – 表示这个 buffer 并未载入，不过，不必担心，载入相当快速的。加号 + 表示这个 buffer 已经修改过了。

:bn buffer next。

:bl buffer last。

- 以上两个指令 elvis 不适用。

:b 档名或编号 移至该档。

- 在 :ls 中就会出示各档案的编号，这个编号在未离开 vim 前是不会变的。这个指令 elvis 也是可以使用。

- 当然 :e#编号 也是可以的，这样的用法则是所有 vi clone 都通用了。
- 如果您是使用 vim 的 GUI，那就在菜单上就会有 Buffers 这个选项，可以很容易的知道及移动各 buffer 间。

:bd(elete) buffer 在未离开 vim 前是不会移除的，可使用这个指

令移除。其实移除她干什么呢？vim 是您在叫用时才会载入的，

因此这些 buffers 并不是像 cache 一般要占内存的。

:e! 檔名 这样也是会开档，但会放弃目前编辑档案的改变，否则

如果档案已有变动，vim 预设是不让您随便离开的。:e! 后不接

什么的话，代表舍弃一切修改，重新载入编辑中档案。

:f 或 Ctrl–g

显示目前编辑的文件名、是否经过修改及目前光标所在之位置。

:f 檔名 改变编辑中的档名。(file)

:r 文件名 在游标所在处插入一个档案内容。(read)

:35 r 檔名 将档案插入至 35 行之后。

gf 这是 vim 的特殊叫档法，会叫出游标所在处的 word 为名的档

案，当然，这个档案要在目前目录内，否则会开新档案。

哦！好像有点给他复杂，主要原因是偶文笔不好啦！不过您何不选个顺手的来用就可以了，选定了，以后就是使用他，这样就不会那么复杂了。:–)

离开

:q 如本文有修改而没存盘，会警告，且无法离开。(quit)

:q! 舍弃所有修改，强迫离开。

:wq 存档后离开。纵使档案未曾修改也是会再存一次档。

:x 也是存盘后离开，但如果档案没有修改，则不会做存盘的动作。

ZZ 和 :x 完全一样，随您高兴用哪一个。

:w 檔名 另存他檔。不加档名就是写入原档。(write)

- :q 及 :q! 是对目前编辑中的档案作用，如果多档编辑的情形并不会离开 vim，这时可下 :qa 或 :qa! 来整个离开 vim。a 就是 all 的意思。
- :指令!，这个 ! 的意思是强迫中止目前正在编辑的动作，而去执行所下的指令。各位应该到目前为止碰过好几次了吧!

vim 的加密功能

vim –x [檔名]

这样进入 vim 后会要求输入密码。以后加密过的档案由 vim 开启时会自动要求输入密码。否则无法开启。其它的编辑器当然是无法开启的。

进入 vim 编辑档案中，临时想加密，可用 :X 指令。

- 小心！vim 一开档就会有个 .檔名.swp 这个档，是为了紧急回复用的，一般是在您所开档案的所在目录，这是个隐藏档，ls 要有 –a 参数才看得到，您加密的功能并没有作用在这个 swp 檔，因此 root 还是知道您在写些什么关于他的坏话的。:–)当然啦！山不转，路转，路不转，人转，您也是可以把 swap 的功能关掉的 :set noswf 就行了！但如果您编辑的是大档案，则不建议您把 swap 关掉，这样会很吃内存的。
- elvis 的话，其暂存档是统一集中存放在 /var/tmp/*.ses，权限是档案所有者始能读写。vim 的早期版本，其 *.swp 档是依原档案的权限来设定的，最近的版本则从善如流，已经改成档案所有人始能读写，就是 –rw––––––– 啦！

紧急 回复

vim –r 档名，或进入 vim 后，下 :recover 档名，来回复。

大家来学 VIM（一个历久弥新的编辑器） [七]

各种标示方法及窗口操作

这个单元多了一种模式，那便是 visual mode（以下简称 v–mode）v–mode 下的反白区（反黑区？^_^）在本文就统一称为标示区，不知各位是否有更好的中文名称？ㄟㄟㄟ，窗口操作和标示有什么关系？为什么摆在这里说明？ㄚ，是因为这两个单元内容都不多，没地方摆，所以就将就凑在一起的啦！乱点鸳鸯谱就请各位见谅。

标示 指令

v 小写 v，这是属于字符标示（character visual），按下 v 后

您就可以移动光标，光标走过的地方就会标示起来。再按一次

v 就会结束 v–mode。您用 mouse 拉出的标示区也是属于这类

的标示。

V 大写 V，这是行标示（line visual），按下 V 后会整行标示起

来（包括行首前空白的部分），您移动上下键，会标示多行。

mouse 连按三次左钮，也是属于此类的标示。再按一次 V 就会结

束 v-mode。

- mouse 按两次左钮是标示一个 word。按三次是整行标示。

Ctrl-v 这是方块标示（block visual），可纵向标示矩形区域。

再按一次 Ctrl-v 就会结束 v-mode。

- 结束 v-mode 的方式亦可使用 Esc 键，或统一使用 Ctrl-c。
- windows 系统下 Ctrl-v 是复制键，可以使用 Ctrl-Q 来替代。

d 删除标示区内容。

y 复制标示区内容。

- 飞…是要先标示好才按的。"ay 还能不能用呢？当然可以，这样就会把标示区内容存于 a 缓冲区中。可以用 "ap 来贴上。

Shift-> 标示区内容向右移一个 Tab。

Shift-< 标示区内容向左移一个 Tab。

您想更深入吗？:h visual.txt 就有详细的介绍，飞，别忘了有 Tab 补全键可以用。

窗口操作

Ctrl-w n 即 :new。开一空的新窗口。

- 这在 vim 会开在原窗口上半方，也就是窗口一分为二。在 elvis GUI 界面的话，则是实实在在的另开一个新窗口（可不是另启动一个 elvis 喔！），当然 elvis 的 console 上也是和 vim 一样，窗口一分为二。而且在 elvis 下，要放开 Ctrl-w 后才按 n，否则 elvis 会不鸟您的。
- 凡窗口操作的按键都是由 Ctrl-w 来起头的，w 就是 window。

Ctrl-w s 即 :sp(lit)，会开一新窗口，且原文件分属两个窗口。

Ctrl-w f 开一新窗口，并编辑光标所在处之 word 为档名的档案。

Ctrl-w q 即 :q 结束分割出来的窗口。

Ctrl-w o 即 :only! 使光标所在之窗口，成为目前唯一显示的窗口

其它窗口会隐藏起来。

Ctrl-w j 移至下窗口。

Ctrl-w k 移至上窗口。

- 还记得 hjkl 的按键移动方式吗?

:sp 文件名 开另一新窗口来编辑档案。

如果您觉得这样分割个窗口来编辑档案不怎么方便，那您可以利用 X 下的虚拟桌面，多开个 xterm + vim 来做多档编辑也是可以的啦！不过这样不仅会多占内存，而且 vim 中好用的书签功能就没法使用了。

大家来学 VIM（一个历久弥新的编辑器） [八]

shell 命令及求助系统

shell 命令

:!外部指令 执行外部指令。当然您的指令要在 PATH 内。

:!! 执行前一次执行之外部指令。

- 在这里的 ! 可不是强迫中止喔!
- vim 中有一个很特殊的指令 @:，是重复前一次的冒号命令。

:sh(ell) 执行 shell。使用 exit 回来。

- 这在 vim GUI 会在原窗口内显示执行结果，在 elvis GUI 则会另开一 xterm 窗口。console 或 xterm 下的 vim 当然就是跳出 vim 进入 shell 中。
- 至于是用哪一种 shell 是可以另外设定的，可由 set shell= 来设定。在 windows 的版本中设定时如遇有空格符档名时要由 \ 来 escape，例如：
-

- :set shell=\"c:\program\ files\unix\sh.exe\"\ –f

  建议抓个 bash 来用啦，配合一些 Win32 中的 UNIX 指令会更适配。

- 在此补充一下补全的功能。不是只有 Tab 键的补全功能喔！也可以使用上下方向键，叫出历史指令，叫出历史指令可用于冒号命令及寻找命令（/）。例如，您前已下了 :!ps aux 这个指令，您可以按 : 后就直接按向上方向键。别忘了！寻找指令也是可以这样用喔！

:r !commond

这个就妙了！会在光标所在处次一行插入外部指令 commond 执行

后的输出内容。例如 :r !date 就会插入日期时间。这在 elvis

是会插入在游标所在处那一行。

:n,mw !commond

以 n 至 m 行内之数据，当做外部指令 commond 的 input。这算

是相当高级的用法了，初学者大概还用不上，不过印象中留有一

个这样的功能，以后总是会用得上的。

K 大写 K 会显示光标所在处之 word 的 manpage。elvis 不适用。

windows 版本亦不适用。

求助系统

原始 vi 是没有 on–line help 的，但 vim 及 elvis 则有相当丰富的说明系统。vim 沿用传统 tag 的方式来找主题，而 elvis 就高明了，是使用 HTML 的格式。原来 elvis 是可以直接阅读 HTML 档的，当然不能显示图文件，但会标明图档的名称。

F1 用过 pe2(3) 吗？好像 F1 是救助键已让大家公认。vim 预设

按 F1 就会叫出说明档。elvis 当然没有，不过您可以自行设

定，在 ~/.elvislib/elvis.rc 中加上

map #1 :help^M

就可以了。须注意的是 ^M 是按 Ctrl–v 后不放再按 M 或

Enter 键，表示马上执行。飞，^M 是特殊字符，是一个字符

，而不是两个字符，您移动光标在 ^M 上就知道了，不是我文

中的，文中的是直接打出来的两字符 ^M。

- 在 console 下要打特殊字符的话，要 Ctrl-Shift-V 然后按您要的键。
-

:h name 这样就会叫出 name 这个说明档，如果后面没接 name，

则会叫出 help.txt 这个总说明档（在 evlis 是 elvis.html）

h 是 help 的缩写。飞，name 记不清楚时，别忘了 Tab 补全。

在 vim 的说明档中，遇有两个 | 围住的主题，把光标移到上面

就可以使用 Ctrl-] 来叫出这个主题的说明，Ctrl-T 可以回到

原说明，elvis 中也是可以这样用。mouse 按两下也是可以叫出

说明。:q 可结束说明档，回到原编辑档案。

:ver 会显示版本、编译信息，编译时加入之参数也会显示出来，

让您知道有加入些什么功能，因为有些功能在编译时就得加入。

其中正号 + 表示有此功能，减号 - 表示无此功能。elvis 只能

显示版本信息，无法显示编译信息。

- 在下都会编译出一个全功能的 vim 来备用，而且 GUI 及 console 下使用的各编一个，以加快 console 下的启动速度。

在此补充说明一点，各位有没有觉得 :q 很常用到，好了！就依 pe2 的习惯设为 F4 吧！怎么设？找上面叫出说明档怎么设，您就依样画葫芦就可以了！vim 的设定档在 ~/.vimrc。个人是直接设成 :q! 以免麻烦，但要记得存盘喔！

大家来学 VIM（一个历久弥新的编辑器） [九]

set 功能设定

本单元可说是 vi(m) 的微调功能，可依您个人的喜好做有限度的调整。由于 vim 做了相当的扩充，文内主要是述叙 vim 的设定，但 elvis 没有的也会标明。但并不打算一网打尽，只说明重要、常用的部份，其它的可以 :h option-list 来查阅。Linux Journal 四月份（April, 2000）有一篇关于 configuring Vim 的文章，有兴趣的朋友也可参考一下，不过依据的版本是 vim 5.5 版，而非最新的 5.6 版。

该在何处设定呢?

可在在线做设定，例如 :set ai 或 :set noai，ai 是 autoindent 的缩写，这样就可以马上改变缩格的设定，但离开 vim 后就又恢复原状。要永久设定就得设在设定档中。 vim 的设定档在：

~/.exrc

~/.vimrc

~/.gvimrc　GUI 版本

$VIM/vimrc

$VIM/gvimrc　GUI 版本

- windows 版本则在 $VIM/_vimrc 及 $VIM/_gvimrc
- 您要把 GUI 的设定设在 ~/.vimrc 也是可以的，但分开来可能对以后要修改时会比较找得到地方改。
- 那 $VIM 在哪里呢? /usr/share/vim 或 /usr/local/share/vim 这是编译时就决定的，但可在 ~/.profie 或 ~/.bashrc 中另设。windows 版本可设在 autoexec.bat 中。

elvis 的则在：

~/.exrc

~/.elvislib/elvis.rc

- windows 版本在 C:\Program Files\elvis\elvis.rc
- 如果您的系统上同时有 vim 及 elvis，则使用 ~/.exrc 要小心，以免互相影响了设定。好处是可以把共同的设定设在 ~/.exrc 里。
- 拜托您一下！设在设定档中时 set 前当然是不必冒号的。:-)

如何 得知目前 的设定

:set 或 :se 会显示所有经过修改的部份，就是和默认值不一样的部份。

:set all 显示目前所有设定值内容。

:set option? 显示 option 这设定的目前值。

:set option 直接在线设定，有些设定需加 = 后加上设定值内容。

:set nooption 取消该设定。

- :set 后面是可以多重设定的。例如 :set autoindent noconfirm autowrite，这样三种设定就会同时重设。

您当然可以改设定档来改变设定值。在 vim 也可以使用 :opt[ion] 来直接在线设定，会列出目前的设定，在 set 这个字上按 Enter 即可改变设定，或就直接修改其值亦可，改完后按 :q 就可以了。在简短说明处按 Enter 则会叫出该部份的说明档给您参考，您说方不方便？改好后

:mk[exrc] 则会写入 ~/.exrc 檔

:mkv[imrc] 则会写入 ~/.vimrc 檔

当然您得先搞清楚您目前所在目录在什么地方，如果您是在家目录启动的那就不用担心了，否则找不到您的新 .vimrc 可不要写信来骂我。:-) elvis 可就没这么方便了，得自行修改 ~/.exrc 或 ~/.elvislib/elvis.rc

各种 set 功能说明

autoindent(ai)

自动缩排，也就是说如果本行是从第五个字符开始写的，您按 Enter

后游标就会停在次行第五个字符处。预设是不打开的。

autowrite(aw)

档案一有更动就会自动存盘。预设不打开。

background(bg)　<vim 才有>

可设成 dark 或 light，这是两种不同的 highlight 颜色设定，详见

$VIMRUNTIME/syntax/synload.vim。不过您要更动颜色的设定，最好

是设在 ~/.vimrc 或 ~/.gvimrc 中，原始档最好不要去动她。

- 飞飞飞，你从没提过 $VIMRUNTIME 好不好！其实这是最近版本的 vim 为了不至安装新版本时把旧版本的一些设定或 macro 档干掉，所以 $VIMRUNTIME 就是 $VIM/vimxx，xx 就是版本号码啦！例如您使用的是 vim 5.6 版，那么就是 $VIM/vim56。

backup(bk)

是否要 backup file。预设不打开。

writebackup(wb)　<vim 才有>

在写入档案前先备份一份，和 backup 的作用不尽相同，请

:h backup-table。预设是打开的，所以您如果不要 backup，那要关

掉的是这个项目，而不是 backup。但请先检查一下您编译时是不是

有编译进去，请 :ver。

backupdir(bdir)　<vim 才有>

设定存放 backup file 的目录。预设在所编辑的档案之所在目录。

binary(bin)　<vim 才有>

设在编辑二进制文件状态，这是防止存二进制文件时把 EOL 也写进二进制

档，那就会悔不当初，如果是图档将会无法再观看，如果是可执行档就

无法执行了！因此预设是 off。

- elvis 会自动判断是否为二进制文件，而且会分成左右两半，左半部会以 16 进位的方式显示，右半部则

是以 ASCII 的方式来显示。

browsedir(bsdir)　<vim 才有>

浏览档案的目录，GUI 版本始有。预设是上一次浏览的目录。就是 GUI

版本菜单上的 [File] –> [Open] 会打开的目录。

cindent(cin)　<vim 才有>

写 C 时很好用，indent 比一般敏感，专为 C 程序代码而设。预设 off。

编辑 C/C++ code 时会自动打开。

cmdheight(ch)　<vim 才有>

状态列的行数，预设一行，建议设成两行。

compatible(cp)　<vim 才有>

设为和原始 vi 兼容的状态，vim 的扩充功能会被抑制。预设 off。

confirm(cf)　<vim 才有>

各种确认动作。预设 off。

directory(dir)

swap 文件存放的目录。前面单元已有说明。

fileformat(ff)　<vim 才有>

这是写入档案时置放 EOL(end of line) 的形式

dos 是以 0D 0A 来断行。

unix 是以 0A 来断行。

mac 是以 0D 来断行。

预设以各系统平台而定，在 Linux 当然是 unix 形式。

fileformats(ffs)　<vim 才有>

可指定多个，会依加载的档案形式来调整 ff。

例如 :set ffs=unix,dos ff=unix

则预设为 unix 格式，但如读入的是 dos 格式的档案，会自动调整

为 dos 格式，这样存档时就会以 dos 格式存盘（状态列会显示）。

。此时如要改成 unix 格式，可 set ff=unix 然后存盘就会转成

unix 格式，反之亦然。

- 如果不这样设，也就是您不管 ff 或 ffs 都设成 unix，那读入 dos 格式的档案时在每行尾会出现 ^M 这个字符（就是 0D 啦！）这时纵使 :set ff=unix 也来不及了！只好 :%s/^M//g 来消去这个 ^M。ㄟ，还记得怎么替换吗？就是把 ^M 换成没有啦！而且 ^M 怎么打出来的还记得吧！翻一翻前面的单元吧！
- Hey，你怎么知道是 0D 呀！好吧！告诉您一个密秘，您把光标移到 ^M 那个位置，然后按 ga 在状态列就会显示 10，16，8 进位的值。其它的字符也是可以如此显示。a 就是 ascii 的意思。但这是 vim 的扩充功能，elvis 没有。
- elvis 纵使加载 dos 格式的档案，也是会自动把 ^M 隐藏起来。

ignorecase(ic)

寻找时不分大小写，这对中文会造成困扰。预设 off。

incsearch(is)　<vim 才有>

加强式寻找功能，在键入 patern 时会立即反应移动至目前键入之

patern 上。预设 off。

hlsearch(hls)　<vim 才有>

寻找时，符合字符串会反白表示。预设 off。如果您是使用 vim 的

预设的 vimrc 档的话，会设在 F8 键来切换。

textwidth(tw)

是一种 word wrap 的功能，从左起算之固定每行的最大字符宽度。

超过此宽度就会自动折行，这可是真的折行，也就是说在折行处会插

入 EOL。预设是 0，也就是没有 word wrap 的功能。

wrapmargin(wm)

和 textwidth 作用相同，只是是从右窗口边向左算起要几个字符起

折行。预设是 0。textwidth 与 wrapmargin 的功能目前并不适用于

中文，打中文还是您自行按 Enter 吧！

wrap

这也是折行功能，可是只是屏幕效果的折行，实际上并没有插入 EOL。

wrapscan(ws)

这和折行没有关系，是指寻找时，找至档尾时，是否要从档首继续找。

预设是要。

paste　<vim 才有>

这是防止在做剪贴时位置会不正确，前面单元已有说明。

ruler(ru)　<vim 才有>

会在状态列显示光标所在处之行列状态，预设不打开，但建议打开。

最右边之代号的意义如下：

Top 档案第一行在屏幕可见范围。

Bot 档案最后一行在屏幕可见范围。

All 档案首尾皆在一个屏幕范围内。

如非以上三种情形，则会显示相对百分比位置。

statusline(stl) <vim 才有>

状态列显示的格式，使用预设就可以了，如果您想骚包一下的话，那

就请您 :h stl。

shiftwidth(sw)

指由 >> 移动整行内容时，一次移动的字符宽度，一般是使用 Tab 的

值，但可由这个设定来改变。

tabstop(ts)

一个 Tab 键宽度。预设是 8 个字符宽度。最好不要随便改，以免您写

的东西由其它编辑器来阅读时造成困扰，为解决这个问题，vim 有一种

softtabstop 的机制，在下一节会详细说明。

showcmd(sc)

在状态列显示目前所执行的指令。

showmode(smd)

在状态列显示目前的模式，例如是 Insert mode 或是 Visual mode。

当然平常的 normal mode(commond mode)是不显示的。

viusalbell(vb)　<vim 才有>

以屏幕闪动代替 beep 声。

number(nu)

显示行号。注意，冒号命令也有 :nu 这是显示光标所在行的行号，您嫌多打一个字的话，:# 也行。不过如果 ruler 打开的话，在状态列本就会显示门前光标所在处的行列值。

list

这也可以算是一种模式，list mode。就是 Tab 的地方会以 ^I 显示，而行尾之 EOL 会显示成 $。可以让您清楚的知道 Tab 在哪里，折行是不是真的。

swapfile(swf)　<vim 才有>

是否需 swap 至磁盘。如果设为 noswf 的话，那将不会有 swapfile 产生，通通会加载在内存中。预设是要 swapfile。

fileencoding(fe)　<vim 才有>

首先先鼓掌一下，啪啪啪…，因为有支持 taiwan，也支持 XIM，也就是说可以使用 xcin-2.5x 来作输入，当然您用 xcin-2.3x 配合 XA 也是可以啦！目前支持简繁中文、日文、韩文，unicode 尚未植入。但前提是您要把 multi_byte 编译进去，这在一开始就讲过了。预设是使用 ansi。set guifont 及 set guifontset 已在一开始讲过，在此就不重复了。

history(hi)

记录冒号命令的历史纪录文件，就是可以用上下方向键叫出来的那锅。

预设是 20 笔。

关于 softt abstop (sts)

几乎所有的 OS 及软件都设定 Tab 就是 8 个字符长，这已经是个公认值，您硬要去改变它的话恐怕带来许多不便，但实际上关于程序风格，许多人又认为 8 个字符太长了，几个巢状循环下来就需折行，反而不方便。因此 vim 体贴您，内建了 softtabstop 的功能，就是由 vim 来代您制造出一个假的 Tab，实际上是空格符组成的 Tab。

举个例子来说明比较清楚。

set softtabstop=4

set shiftwidth=4

这样会由 4 个空格符取代一个 Tab，您按 Tab 键 vim 就跳 4 格，需注意的是，如果您按了三次 Tab 键，那就是一个实际的 Tab 加上四个空格符，可不是 12 个空格符喔！是混合 Tab 及 space 的。

问题来了！那我要按真正的 8 字符的 Tab 时怎么办？简单，还记得怎么按特殊字符吗？Ctrl–v Tab 或 Ctrl–v I 就可以了，那就是如假包换的 8 字符长之 Tab。当然，您按两次 Tab 不就得了！:–)

关于 折行

前面已说过 set wrap 就可以造成屏幕折行，可是却会把一个英文单字折成两半，实在很不雅观。好了，vim 再体贴您一次，set linebreak(lbr) 就会避免这种问题发生，会在空白或标点符号的地方来折行，但也仍属屏幕折行，并不会插入 EOL。这个功能目前在中文判断上还是会出槌！:–(

大家来学 VIM（一个历久弥新的编辑器） [十]

规则表示式的运用

在本系列文章一开始就说明了学 vi(m) 可以顺便学规则表示式（regular expression，以下简

称 regexp），那为什么到现在才来讲呢？因为 regexp 说简单也算不很难，但您要深入去使用的话，有时会马上看不出一个复杂的 regexp 在说些什么的，就曾有人形容 regexp 为「有字天书」！ 而且在 vi(m) 整体都还没一个概念就加入 regexp 的话，那后面的单元恐怕就没人看了！而 regexp 各家有各家的 extensions，这也是大家视为畏途的原因之一，不过总是大同小异，只需注意一下就可以了。目前先不必管别家怎么说，就让 vim 暂时先成为我们的「标准」，以后碰到其它程序的 regexp 应该就可以触类旁通。以下我们尽量由实例去了解。当然，小小的一篇文章是没有办法详尽介绍，只能捡重点来说明了。如有疑问，可 :h pattern 或在 Un*x 系统中可 man 7 regex，甚至 man ed，man sed，man grep，man awk，man perlre 里面也是会说些 regexp，但要注意和 vim 差异的地方！其中 perl 的 regexp 应该是最完整的了，如果您的系统没有 perl 那应该是「稀有动物」了！:-) 飞飞飞！ vim 只是一个编辑器，可不是独立的程序语言！

基本的匹配

* 指前所绑住的字符或字符集合，出现 0 次或 0 次以上。

\+ 和 * 作用相同，但不包括出现 0 次。

\= 指前所绑住的字符恰好出现 0 或 1 次。

\| 这是多选，就是 or 的意思，被 \| 隔开的 pattern，任一个符

合的话就算符合。

- \+, \=, \| 会加上一个 \，是因原字符在 vi(m) 就具有特殊意义，在一般的 regexp 中是 +，?，| 就可以了，只是提醒您一下，以免搞混了！
- 记住 * \+ 是不可数的！用辞不是是精确，只是帮助您记忆啦！
- 在 elvis 及 ed 中是使用 \? 来匹配出现 0 或 1 次，而不是 \=，这里要非常小心！

[实例] dg*
指 * 前所绑住的字符 g 出现 0 次或 0 次以上。也就是说 d(出现 0 次)，dg，dgggg，dgggggggg 都是符合这个 pattern。如果您下寻找指令 /dg*，那符合这个 pattern 的字符串都会被找出来。如果用在代换就要非常小心了，像 extended 中的 d 也是会被置换掉的。例如您下 :%s/dg*/test/g 的话，那 extended 这个字会换成 extentestetest。

- shell 中使用的通用字符为 pattern matching notation 和 regexp 不同的。dg* 在 shell 中是解为以 dg 开头的任意字符串，这就不包括 d 在内了，也就是说在 shell 中，* 是代表任一字符或字符串。

[实例] dg\+
dg, dgg, dgggggg 皆符合，但 d 则不符合。如果是 dg\= 的话，就只有

d、dg 这两个符合了。

[实例] :%s/The\|All/test/g
全文中只要是 The 或 All 都会被替换成 test。注意，如果文中有 There 也是会被替换成 testre！要如何避免这种情形呢？下面会另述及限定使用法。

[实例] /123-\=4567
这样会找出，123-4567 及 1234567。当然 123-456789 也是会被找出来。

[...] 字符集合，表示中括号中所有字符中的其中一个。

[^..] 这是上述 [...] 的补集，表非中括号内字符的其中一个。

. 除换行字符外的任一单一字符。指本身，非指前所绑之字符。

就好像 shell 中的 ? 一样。如果要指定真正的英文句点，要

用 \ 来 escape，就是说 \. 这时的 . 是代表真正句点，而不

是 regexp 中的特殊意义。其它如 * 亦同。

[实例]

[Aa]

A 或 a 其中的一个。

[12345]

12345 其中的一个数目字。可用 [1-5] 来表示。连续性的数目字或字符可用 - 来隔开，写出头尾来代表就可以了。[0-9] 就表 0 到 9 的数目字，[a-d] 就代表 abcd 四个英文字母

[实例] W[0-9]*\.cc
这个例子是说以 W 开头，后接 0-9 其中一个或多个数目字或不接什么，然后是一个句点，最后是 cc。所以
W.cc，W1.cc，W2.cc，W345.cc，W8976543287.cc 皆符合。如果要表示 W 及 .cc 间夹一个以上的数目字，要写成 W[0-9][0-9]*\.cc。

[实例] .*
这代表任意字符或字符串，或什么都没有，脑筋急转弯，对照前面的定义想一下。当然这是不包括换行字符的。

[实例]
[^M]　表除 M 以外的任意字符。
[Tt]　表 T 及 t 以外的任意字符。
[^0-9]　表非数目字之字符。
[^a-zA-Z]　表非英文字母之字符。

- 注意，^ 要在中括号内，且在最开头的地方，否则另有含意。

^ 匹配行首，指其后绑住的字符串，出现在行首才符合。

$ 匹配行尾，指其前绑住的字符串，出现在行尾才符合。含换行字符。

- 不是在行首的 ^ 指的是 ^ 这个字符。不是在行尾的 $ 是指 $ 本身这个字符。

[实例] /^What
这样只有在行首的 What 才会被找出来。注意！ Whatever, What's 也是会被找出来。如果是 /What$ 则是在行尾的 What 才会被找出来。

[实例] ^$
这是什么东东？行首也是行尾的行。丫，就是空白行嘛！当然也不能说这个行是没有什么东东啦！空白行至少也是会有个换行字符。在后面会详述如何消除全文的空白行。

\(...\)　记忆 pattern，可由 \1, \2...\9 来叫出。

[实例] :%s/\([a-z]\)\1/test/g
这样 aa, bb, cc, dd, ..., zz 都会被 test 替换掉。这和 :%s/[a-z][a-z]/test/g 是不一样的意思，后者会把 aa, ab, ac... ba, bb, bc...zz 都换成 test。也就是说 \(...\) 由 \1 叫出时会有对称性的配对出现。

[实例] :%s/\(.\)\(.\)r\2\1/test/g
会将中间为 r，前有二个任一字符，后有两个具对称性的字符所组成的字符串替换成 test。\2 是呼叫第二组 \(.\)，而 \1 是呼叫第一组 \(.\)。例如：12r21，cfrfc，7grg7 等都会被替换成 test。

\<　匹配字（word）首。所谓 word 包括文数字及底线。

\>　匹配字尾。这就是前所提及的限定用法，被 \<，或 \> 括住的

pattern 就会被限制住，使 regexp 不能再向右（左）扩充解释。

- ed 及 perl 中可以 \b 来表示这两个符号，perl 中只支持 \b，ed 则 \b 及 \<, \>皆支援。但在 perl 可多加个 ? 来限制 regexp 的扩充解译。
- 功能上而言，这是和 ^ $ 一样的定位样式（anchor pattern）指所绑住的字符串必须是单字边界（word boundary），前或后或前后除了空格符及标点符号外不可再有其它字符。
- 在 vim 中 \b 是表示 <BS> 即 backspace 键。

[实例] :%s/\<abbbc\>/test/g
这样只有 abbbc 才会被替换成 test。如果没有这样限定，:
%s/abbbc/test/g，那 deabbbcly 中的 "abbbc" 亦会被替换成 test。所以前面 :%s/The\|All/test/g 可换成 :%s/\<The\>\|\<All\>/test/g 这样一来，There 就不会被替换成 testre 了!

[实例] :%s/\<abbbc/test/g
这样的话，只要是以 abbbc 为首的字(word)，其中的 abbbc 的部份都会被 test 所替换。注意！是指前缀，而不是指行首。所以
abbbc，abbbcerd，abbbckijuds 都符合。

\{n,m} 指前所绑住的字符或字符集合最少出现 n 次，最多出现 m 次。

- 这在一般的 regexp 表示成 \{n,m\}。vim 及 elvis 两种表示法皆支持。perl 则直接使用 {}。以下会举四种不同的例子，请大家发挥一下想象力。:-)

[实例] \{最小值，最大值}
如 [0-9]\{3,4} 匹配至少三位数，但不可多于四位数的数目字。如：

123

12

1

123456

1234567

12345678

1234

12345

如果下 :%s/[0-9]\{3,4}/test/g 的话，那 1，12 这两组不会被替换，因为不

满 3 位数。而 12345，则会换成 test5。123456，则会换成 test56。12345678，则会换成 testtest。1234567 也是会换成 testtest。123，1234 这两组则会被替换成 test。您可以亲自操作一次就知道怎么一回事了。操作时最后加 gc 来 confirm，这样您会更了解实际替换的内容。飞，别忘了 u 可以回复您的编辑动作。

[实例] \{数目字}
xy\{20}　表示 x 后接 20 个 y。
e[x-z]\{4}　表示 e 后接有四个字符，是 x,y,z 的其中一个的组合。如：exxxx, exyyz, ezzyz, exyzz 皆符合。

[实例] \{最小值，}
xy\{2,}　表 x 后接至少二个的 y。相当于 xyyy* 或 xyy\+ 。

[实例] \{，最大值}
xy\{,4}　表 x 后接至多四个或更少的 y（可能没有）。因此 x, xy, xyy ,xyyy, xyyyy 皆符合。

中介字符（metach aracte r, o r cha racte r classes ）

主要是简化 regexp 的书写。

\s 表空格符，即 <Space> 或 <Tab>。

- 不含换行字符，这是编辑器的特性使然。在 perl 的 \s 是包含换行字符的。而且 vim 及 elvis 皆不支持 \n 这种换行中介字符。

\S 表非空格符。

\d 表数目字（digits），即 [0-9]。

\D 表非数目字，即 [^0-9]。

\w 表一般字符（word character），包括底线。即 [0-9a-zA-Z_]。

\W 表非一般字符，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 表小写字母（lowercase character），即 [a-z]。

\L 表非小写字母，即 [^a-z]。

\u 表大写字母（uppercase），即 [A-Z]。

\U 表非大写字母，即 [^A-Z]。

- 原始 vi 不支持此种中介字符。
- 使用中介字符的比对速度将会比使用字符集合 [] 的快。

全域性的指令

:[range]g/pattern/[cmd]

cmd 是 ed 可用的指令，预设是 p(print)，您可查一下 man ed，就可以知道有什么指令可用。这个小节里主要是说明 d(delete) 的功能。因为是要说明如何消除空白行。需注意的是，d 是行删除指令，凡含 pattern 的整行都会被删掉，而且 range 不指定的话，预设是全篇文章，因为 g 就是代表 globe。

- 在 vim 的 help 文件里说的是 ex 指令，但 ex 实际上是和 vim 连结的，因此这里特别指出 ed。但 ed 的指令少数可能会和 vim 的 ex 不同，这是因为 ed 和 vim 并异步在发展，作者也非同一人。

:g/^$/d

这样就会删除全文的空白行。前面已提过 ^$ 代表的是空白行。但这里有个问题，如果空白行里包含了其它空格符（即 Space 或 Tab）的话。表面看起来是和一般空白行一模一样，但却暗藏玄机，用上面的方法就无法删除这种空白行了！怎么办？来！看招！

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

在 vim 或 elvis 里您可以如此照打，也就是 <Space> 代表空格符，<Tab> 代表按 Tab 键的结果。在原始 vi 则不行，得自行按出特殊字符出来，就是 Ctrl-v Space 及 Ctrl-v Tab。或采更简单的打法：

:g/^\s*$/d

还记得中介中元吗？好用吧！少打了不少字。:-) 意思就是删除含 0 或 1 个以上空格符的行。

有些书中写成 :%s/^$//g 可以删除空白行，这是错误的，因为 :s 这 个指令只更动一行里的内容物，但不会做删除一行的动作。

& 替代变数

代表置换时合于 patern 的字符或字符串。

[实例] :%s/\u\d\d\d\d\d\d\d\d\d\>/ID:&/g
这样全文中的身份证字号前就会加上 ID: 字样，也就是说 T123456789 会被换成 ID:T123456789。还记得吗？ \d 就是 [0-9]，\u 代表大写的英文字母。加个 \> 是防止 T12345678999 也被换掉。当然前面再加个 \< 更保险。
ID: 字样您用中文也行！
另一个好用的例子是电话号码前加上 TeL:，就请您自行练习了！

[实例] 将档案 3 至 7 行的数据向右移 2 个空白

```
:3,7s/.*/  &/
```

但这样连空白行也是会插入空格符，较高明的做法是：

```
:3,7s/.\+/  &/
```

这样空白行就不会去动它了！想通了 .* 及 .\+ 的意思了吗？往前翻一下 . * \+ 的定义。

[实例] 将档案 3 至 7 行的数据向左移 2 个空白

```
:3,7s/^  //
```

就是删去行首的二个空白啦！

[实例] 将全文的 Edward 这个单字，前后加上中括号

```
:%s/\<Edward\>/[&]/g
```

[实例] 将全文的 Edward 这个单字，改成大写的。

```
:%s/\<Edward\>/\U&/g
```

- 乀！\U 不是代表非大写字母吗？喔！您搞错位置了。\U 在 pattern 的位置的时候是指非大写字母的样式，即 [^A-Z]，但如果是在置换字符串位置的时候是指将其后的字符串通通改成大写。与其相对的是 \L，会将其后的字符串改为小写。详细请 :h sub-replace-special。

[实例] 将全文每行最后加上
 这个 HTML tag。

```
:%s/.*/&<BR>/g
```

怎么样，是否已感觉到 regexp 威力无穷了呢？还是您已经快睡着了呢？:-) 不过也请您想想，如果是在没有 regexp 功能的编辑器里，范例中的一些动作您会怎么做呢？一个一个去改？

gre edy 陷阱

regexp 会有贪心的倾向，什么意思呢？就是说在同一行内，如果有多个符合 pattern 的情形，会找最长的那一个。

- 注意！greedy 的特性是针对会反复比对的 regexp 而言，例如：*, \=, \+, \{} 等。前面所举的 .* 的例子，由于 greedy 的关系，在整篇文章中做替换时，会被当成是每一行整行，因为 regexp 会去找每一行最长符合的那一个。

[实例] 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 换成 program 的，结果由于 regexp 的贪心，整个 "This is a test. Test" 会换成 program。结果原文就变成了 program for regexp. 因此在全文替换时要非常小心，避免使用弹性太大的 regexp。像此例，只要下 :%s/\<[Tt]est\>/program/g 就可以了！

最后提醒您，这可不是 regexp 的全部，碍于篇幅及在下功力的问题，当然是没办法全面详尽的向各位做介绍，在下只是将各位领进门，修行就得看各位了！如果还想更深入的研究 regexp，可参考： Mastering Regular Expressions(O'Reilly & Associates) 一书。