**vim note**

1. Vundle introduction
   1. [Vundle](https://github.com/gmarik/Vundle.vim) 是一個可以**自動下載**、**安裝**、**更新**與**管理**的工具，讓 Vim 的使用者可以很方便的使用各種 plugins。
   2. Vim 編輯器可以透過各種 plugin 來增加各種功能，在 [Vim Scripts](http://vim-scripts.org/vim/scripts.html) 網站上收錄了非常大量的Vim指令稿，使用者可以自己下載後安裝在Vim中使用。雖然這些plugins可以加強Vim的功能，但是如果安裝了太多的 plugins，在管理上就會比較麻煩，這時候就可以使用 Vundle 來幫忙管理所有的 Vim plugins。
   3. **Plugin & Bundle 指令**
      1. 目前Vundle已經把指令都改成Plugin開頭,但仍支援Bundle指令, 例如 : **Plugin** 'taglist.vim'
      2. 仍可以使用，**Bundle** 'taglist.vim' ,但是不建議
   4. vim在homebrew的更新指令，此指令會將目前系統的vim給覆蓋掉，然後裝新的vim(目前更新到8版)

brew install vim --override-system-vim --------> old

brew install vim --with-override-system-vi ----------> new

* 1. 清除原本的vim(表示系統內已用brew去裝了vim)，然後安裝有python3的vim:

brew remove vim

brew cleanup

brew install vim --with-python3

* 1. **強制清除vim**
     1. brew uninstall --force vim (還未試過可以試試看)
  2. continue

1. 安裝Vundle
   1. 執行

git clone https://github.com/VundleVim/Vundle.vim.git ~/.vim/bundle/Vundle.vim

* 1. continue

1. 查看在Vundle中安裝了哪些套件
   1. 執行 :PluginList
2. 安裝所有的套件
   1. 增加 ”.vimrc”內的Plugin item，然後儲存並且離開vim。
   2. 重新進入vim，執行 :PluginInstall
   3. restart vim
3. 安裝單一套件
   1. 增加 ”.vimrc”內的Plugin item，然後儲存
   2. 並且在vim系統內執行 :PluginInstall “套件名稱”
   3. restart vim
4. remove Plugin
   1. 刪掉 ”.vimrc”內的Plugin item，然後儲存並且離開vim。
   2. 重新進入vim，執行 :PluginClean
   3. restart vim
   4. continue
5. 刪除單一套件
   1. 執行 :PluginClean “套件名稱”
   2. Restart vim
6. 更新Plugin
   1. 執行 :PluginUpdate -->所有已安裝的套件將會更新.
   2. Restart vim
7. 搜尋Vundle上所有的套件
   1. 執行 :PluginSearch
   2. 搜尋單一套件 :PluginSearch 套件名稱
8. .vimrc的設定說明
   1. YouCompleteMe
      1. 安裝方法
         * 在.vimrc中加入Plugin ‘Valloric/YouCompleteMe’, 儲存然後離開.
         * 重新進入vim, 執行PluginInstall, 去安裝該套件.
         * 執行下面指令, 可以進入YouCompleteMe資料夾,

cd ~/.vim/bundle/YouCompleteMe

* + - * 執行下面指令, 可以編譯YouCompleteMe套件, 在編譯的選項裡設定包含C語言的補全功能,

./install.py --clang-completer

* + - * continue
    1. .vimrc的設定

" YouCompleteMe --------------------------

1. **" 默認配置文件路徑"**

let g:ycm\_global\_ycm\_extra\_conf = '~/.ycm\_extra\_conf.py'

1. **" 打開VIM時不再詢問是否加載ycm\_extra\_conf.py配置"**

let g:ycm\_confirm\_extra\_conf=0

set completeopt=longest,menu

1. **" python3直譯器路徑"**

let g:ycm\_path\_to\_python\_interpreter='/usr/local/anaconda3/bin/python3'

1. **" 是否開啟語義補全"**

let g:ycm\_seed\_identifiers\_with\_syntax=1

1. **" 是否在註釋中也開啟捕全"**

let g:ycm\_complete\_in\_comments=1

let g:ycm\_collect\_identifiers\_from\_comments\_and\_strings = 0

1. **" 最小自動觸發補全的字元大小設置為2"**

let g:ycm\_min\_num\_of\_chars\_for\_completion=2

1. **" 補全後自動關閉預覽視窗"**

let g:ycm\_autoclose\_preview\_window\_after\_completion=1

1. **" 禁止緩存匹配項，每次都重新生成匹配項"**

let g:ycm\_cache\_omnifunc=0

1. **" 字串也開啟補全"**

let g:ycm\_complete\_in\_strings = 1

1. **" 離開插入模式後自動關閉預覽窗口"**

autocmd InsertLeave \* if pumvisible() == 0|pclose|endif

1. **"回車即選中當前項"**

"inoremap <expr> <CR>       pumvisible() ? '<C-y>' : '\<CR>'

1. **"上下左右鍵行為"**

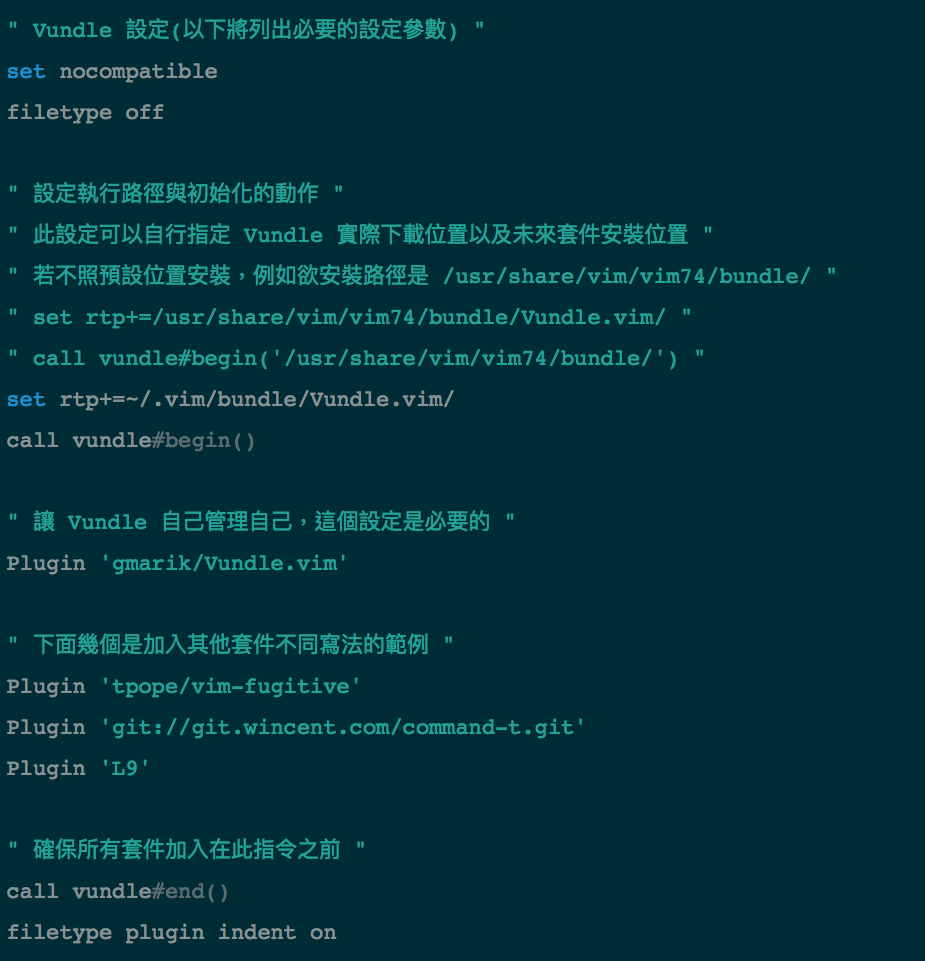
inoremap <expr> <Down>     pumvisible() ? '\<C-n>' : '\<Down>'

inoremap <expr> <Up>       pumvisible() ? '\<C-p>' : '\<Up>'

inoremap <expr> <PageDown> pumvisible() ? '\<PageDown>\<C-p>\<C-n>' : '\<PageDown>'

inoremap <expr> <PageUp>   pumvisible() ? '\<PageUp>\<C-p>\<C-n>' : '\<PageUp>'

* + 1. continue
  1. continue
  2. other



1. F5編譯的設定

map <F5> :call CompileAndRun()<CR>

func! CompileAndRun()

exec "w"

if &filetype == 'c'

exec "!gcc -std=c11 % -o /tmp/a.out && /tmp/a.out"

elseif &filetype == 'cpp'

exec "!g++ -std=c++11 % -o /tmp/a.out && /tmp/a.out"

elseif &filetype == 'java'

exec "!javac %"

exec "!java %<"

elseif &filetype == 'sh'

:!%

elseif &filetype == 'python'

exec "!python3 %"

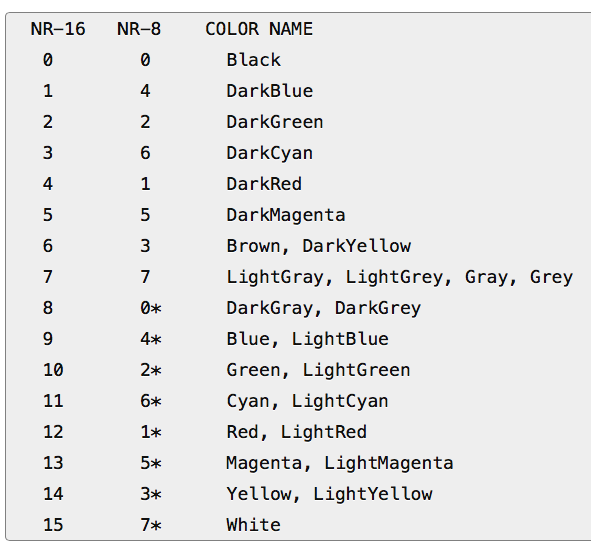
endif

endfunc

1. 挑選 Vim 顏色(Color Scheme)
   1. 下載 COLOR\_SCHEME.vim
   2. mkdir ~/.vim/colors/
   3. mv COLOR\_SCHEME.vim ~/.vim/colors/
   4. 在vim ~/.vimrc裡, 加入下述內容

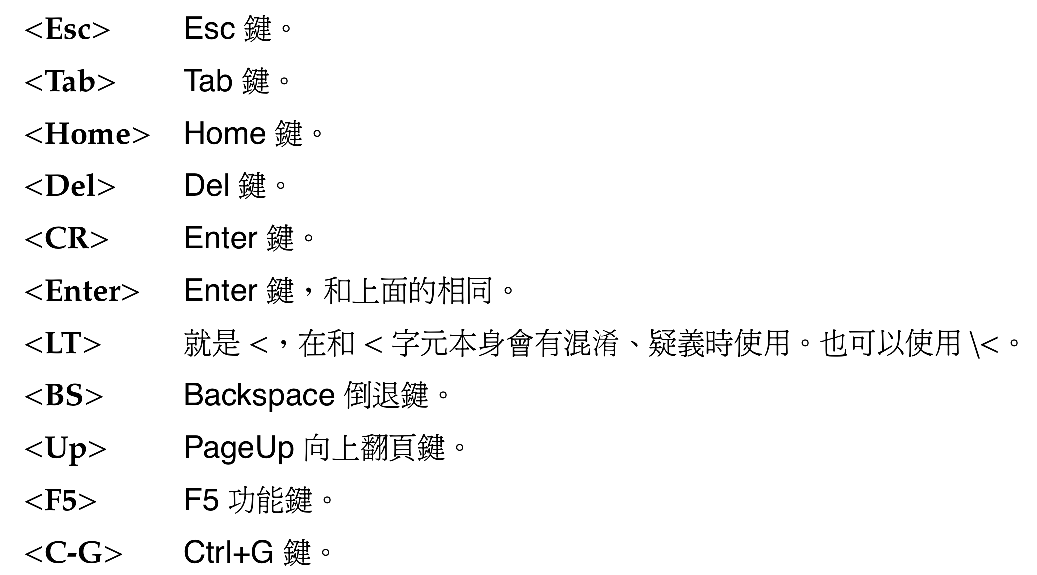
*colorscheme COLOR\_SCHEME*

1. type of vim mode
   1. **1: normal** mode --> ESC
   2. **2: insert** mode --> i
   3. **3: command line** mode --> :
   4. **4: search** mode --> / or ?
   5. **5: visual** mode --> v
   6. **6: visual** **line** mode --> V
   7. **7: visual** **block** mode(column mode) --> ctrl-v
2. 指令用法
   1. 取代string :
      1. 先按一下「Esc」, 輸入「**:1,$s/123/456/g**」
      2. 再按「Enter」即可作搜尋並取代的動作!
      3. 此範例是將文字**123**更換成文字**456**，並且將此文件裡的所有符合的皆取代。
      4. **1,$** 指的就是從頭至尾,
      5. s(substitute)替代。
      6. 在比對(match)的時候，如果不加「g」這個額外選項的話， vim 只會把每一行比對到的第一個作取代，同行其他也 match 到的就不管了。所以用「g」表示每一行中每一個比對到的都要置換。
      7. :起始行,終止行s/搜尋字串/取代字串/gic --> 從第 n 行到第 n 行取代字串 (g: 整行全部 i: 不分大小寫 c: 詢問)
      8. :1,$s/搜尋字串/取代字串/gic --> 全部取代字串 (g: 整行全部 i: 不分大小寫 c: 詢問)
      9. continue
   2. 標記與複製
      1. yy: 複製整列的資料
      2. 2yy: 複製2列的資料
      3. yy2: 複製同一列的資料2次
      4. p: 在**下**一行貼上複製或刪除的內容
      5. P: 在**上**一行貼上複製或刪除的內容
   3. 搜尋與取代
      1. / --> 向**下**搜尋字串
      2. ? --> 向**上**搜尋字串
      3. :set ic --> 搜尋時不分大小寫
      4. :set noic --> 搜尋時要分大小寫
   4. 刪除
      1. **dd** --> 刪除游標整列的資料
      2. **數字＋dd** --> 刪除 n 行
      3. dG --> 刪除游標列到最後一列
      4. d1G --> 刪除游標行到第一行
      5. **d$** --> 刪除游標處到最後一個字元
      6. **d0** --> 刪除游標處到第一個字元
      7. continue
   5. 檔案功能
      1. :w --> 存檔 (加 ! 表示強制存檔)
      2. :w 檔名 --> w後面加入檔名, 可以將該檔案以此名稱儲存.
      3. :files --> 顯示所有已經打開的檔案
      4. :Ex --> 開啟檔案瀏覽器
      5. :Hex --> 分割水平視窗後，再開啟檔案瀏覽器
      6. :**Vex --> 分割垂直視窗後，再開啟檔案瀏覽器**
      7. **continue**
   6. 視窗分割
      1. **vnew --> 新增垂直分割視窗**
      2. :vsp --> 新增垂直分割視窗，並在新增的視窗載入檔案
      3. **[Ctrl]-W → [方向鍵] --> 切換視窗**
      4. continue
   7. 頁籤
      1. gt --> 移至下一個頁籤
      2. :tabs --> 列出所有頁籤
   8. Other command
      1. u --> 還原指令
      2. [Ctrl]-r --> 前進指令
      3. . --> 重複上一個指令
      4. J --> 將游標行與下一行合併
   9. 檔案瀏覽器操作 : 先以Ex進入檔案瀏覽頁面
      1. - --> 到上一層目錄
      2. d --> 建立目錄
      3. D --> 刪除目錄
      4. R --> 重新命名
      5. v --> 新增垂直視窗
      6. continue
   10. continue
3. set項目
   1. :Set number --> 顯示列號
   2. :syntax on --> 語法高亮度顯示
   3. :set hlsearch --> 標記搜尋到的字串
   4. :set autoindent --> 自動縮排
   5. :set ruler --> 底部的行號等顯示
   6. :set tabstop=4 --> 設定 tab 鍵的字元數
   7. :set expandtab --> 輸入 tab 都會被轉換成 space，而對應的 space 數量由 :set tabstop=x 決定
4. Color
   1. highlight Comment ctermfg=cyan --> 設定註解的顏色
   2. highlight Search term=reverse ctermbg=4 ctermfg=7 --> 設定搜尋到的字串顏色
   3. 行號顏色
      1. :hi LineNr cterm=bold ctermfg=DarkGrey ctermbg=NONE
         * 設定行號為：粗體，前景色為深灰色，沒有背景色
         * hi 為 highlight 指令的縮寫，擁有同樣的效果
         * LineNr 指定為 line number 的顏色設定
         * cterm 指的是爲 color terminal 的環境 ( 在 linux 桌面環境所開的 terminal ) 做設定，其他還有 term ( 一般 terminal，例如透過 putty 連上的工作環境 )、gui。
         * cterm 支援的顏色列表，一共有 16 色

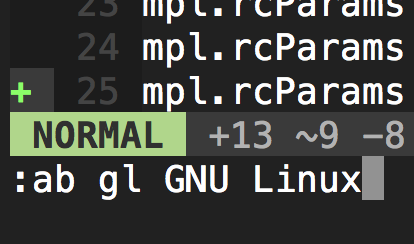


* + 1. :hi CursorLineNr cterm=bold ctermfg=Green ctermbg=NONE
       - hi CursorLineNr cterm=bold ctermfg=Green ctermbg=NONE
       - **設定游標所在的行號為：粗體，前景色為綠色，沒有背景色**
       - 配合 :hi LineCr 就可以很清楚的知道現在游標所在的位置
       - continue
    2. continue
  1. continue

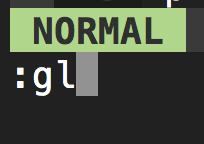
1. Vim script
   1. 按鍵對應
      1. V*im* 可以將多個動作(命令或是函數)對應給一個簡單的按鍵
      2. V*im* 預設已經內 建把 F1 鍵定義給線上求助檔，因此，這個鍵就不要去麻煩他了。
      3. 按鍵的書寫方法



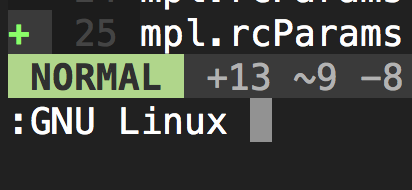
* + 1. 查閱vim裡面按鍵的對應寫法 --> :h key-notation
    2. continue
  1. map 指令的種類
     1. nmap(Normal mode): 僅在常態模式時作用
     2. imap(insert mode): 僅在插入模式時作用
     3. cmap(command line mode): 僅在命令列模式時作用
     4. vmap(visual mode): 僅在反白模式時作用
     5. omap(operator-pending mode): 僅在操作等待模式時作用
  2. map的寫法範例：
     1. 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。
     2. Ex: map <F5> bi{<Esc>ea}<Esc>
        + 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。
     3. Ex: map ,b bi{<Esc>ea}<Esc>
        + 按 “,b”
        + 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)。
     4. continue
  3. 防止重複對應
     1. 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
     2. nore type
        + nnoremap => Normal
        + inoremap => Insert
        + cnoremap => Command-line
        + vnoremap => Visual
        + onoremap => Operator-pending
        + noremap => Normal, Visual and Operator-pending
  4. 縮寫對應
     1. 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**。
     2. 只能用下列mode
        + 插入模式、
        + 取代 模式及
        + 命令列模式
     3. Ex: ab gl GNU Linux
        + ab 是 abbreviate 的縮寫. As following Pic



* + - * Into command line mode, input gl as following pic



* + - * After imputing “gl” then press “**space**” button as following pic



* + - * continue
    1. 常會用到的指令: **“””look here”””**
       - :ab[breviate] --> 不接任何參數，這會列出目前所有的縮寫對應。
       - :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。
    2. continue
  1. Vim script 只有兩種資料型態，**數字**及**字串**。
  2. key mapping
     1. :inoremap ( ()<Esc>i --> 在insert-mode下，只要輸入 ( ，vim會先執行輸入 () → <Esc> 進入 Command-mode → i 進入 Insert-mode → 就會發現游標在 “(” “)” 之間了
     2. continue
  3. continue

1. leader
   1. continue
   2. 看leader vaule的方法：
      1. :nmap foo :echo('your leader is "<Leader>"')<Esc>
      2. Input “foo” string then show leader’s definition.
      3. 如果以前沒有設置，它可能是未定義的，而是默認為反斜線\
   3. continue
2. filetype
   1. filetype indent on -->
      1. 啟用依照檔案類型，決定自動縮排樣式的功能
      2. 例如：如果正在編輯的檔案是 .c 或 .cpp 檔，遇到 { 按下 enter ，會自動往右一縮排。如果是 Makefile，輸入完 target 按下 enter ，為了 rule 而自動往右一縮排。
3. Plugin 套件說明
   1. 'scrooloose/nerdcommenter'
      1. 註解一整列 --> <leader>cc
      2. 刪除註解一整列 --> <leader>cu
   2. continue
4. continue