**Topic**

1. **vim-plug套件**
2. **vim type of mode**
3. **vim operation**
4. **.vimrc setting**
5. **self package setting**
6. **external package setting**
7. **vim顏色編輯**
8. **continue**
9. **vim-plug套件**
10. vim-plug是一個輕量級管理vim套件的工具.
11. 在.vimrc裡面加入以下的code可以自動安裝該套件(如果系統還未安裝的話)
    1. For mac & unix

" Install vim-plug if it's not already installed (Unix-only).

if empty(glob('~/.vim/autoload/plug.vim'))

silent !curl -fLo ~/.vim/autoload/plug.vim --create-dirs

\ https://raw.github.com/junegunn/vim-plug/master/plug.vim

autocmd VimEnter \* PlugInstall --sync | source $MYVIMRC

endif

* 1. For mac & unix & win

" Download and install vim-plug (cross platform).

**if empty(glob(**

**\ '$HOME/' . (has('win32') ? 'vimfiles' : '.vim') . '/autoload/plug.vim'))**

**execute '!curl -fLo ' .**

**\ (has('win32') ? '\%USERPROFILE\%/vimfiles' : '$HOME/.vim') .**

**\ '/autoload/plug.vim --create-dirs ' .**

**\ '**[**https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim**](https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim)**'**

**autocmd VimEnter \* PlugInstall --sync | source $MYVIMRC**

**endif**

1. Vim-package 放置的位置

" Manage plugins with vim-plug.

call plug#begin()

**--套件放置在這之間--**

call plug#end()

1. Vim-plug command
   1. PlugInstall -> 安裝套件, 後面沒加名稱可安裝全部的套件。
   2. PlugClean -> 刪除套件
   3. PlugUpdate -> 套件更新
   4. PlugUpgrade -> vim-plug管理軟體更新
   5. PlugStatus -> Check the status of plugins
   6. PlugDiff -> 檢查上次更新的更改和有待的更改
2. .vimrc儲存並且重啟(reload)
   1. :w | source $MYVIMRC
3. Package安裝程序
   1. 編輯.vimrc檔案, EX:

call plug#begin() " Manage plugins with vim-plug.

Plug 'ctrlpvim/ctrlp.vim'

Plug 'easymotion/vim-easymotion'

Plug 'mileszs/ack.vim'

Plug 'scrooloose/nerdtree', { 'on': 'NERDTreeToggle' }

Plug 'tpope/vim-unimpaired'

Plug 'tpope/vim-vinegar'

call plug#end()

* 1. 儲存並關閉.vimrc檔案然後重啟.vimrc(亦可用指令:w | source $MYVIMRC)
  2. 執行PlugInstall 指令
  3. 儲存, 關閉, 重啟, 安裝一起, 執行指令 :w | source $MYVIMRC | PlugInstall

1. Package刪除程序
   1. 編輯.vimrc檔案, EX:

call plug#begin() " Manage plugins with vim-plug.

Plug 'ctrlpvim/ctrlp.vim'

Plug 'easymotion/vim-easymotion'

Plug 'mileszs/ack.vim'

Plug 'scrooloose/nerdtree', { 'on': 'NERDTreeToggle' }

Plug 'tpope/vim-unimpaired'

Plug 'tpope/vim-vinegar'

call plug#end()

* 1. 儲存並關閉.vimrc檔案然後重啟.vimrc(亦可用指令:w | source $MYVIMRC)
  2. 執行PlugClean 指令

1. Package 更新
   1. .vimrc檔案開啟
   2. 執行:PlugUpdate
2. Vim-plug軟體更新
   1. 打開.vimrc檔案
   2. 執行:PlugUpgrade
3. **vim type of mode**
   1. continue
   2. continue
4. **vim operation**

狹義word: 字串由\_, 字母, 數字所組成.

廣義word: 狹義word以外的字串.

* 1. 方向鍵
     1. h -> left
     2. j -> down
     3. k -> up
     4. l -> right
  2. 編輯復原 – 後退or 前進
     1. 後退 -> u
     2. 前進 -> Ctrl + r
  3. 視窗
     1. 視窗切割

sp -> 水平切割視窗

vs -> 垂直切割視窗

Ctrl + h -> 向左跳

Ctrl + j -> 向下跳

Ctrl + k -> 向上跳

Ctrl + l -> 向右跳

* + 1. 改變視窗的大小

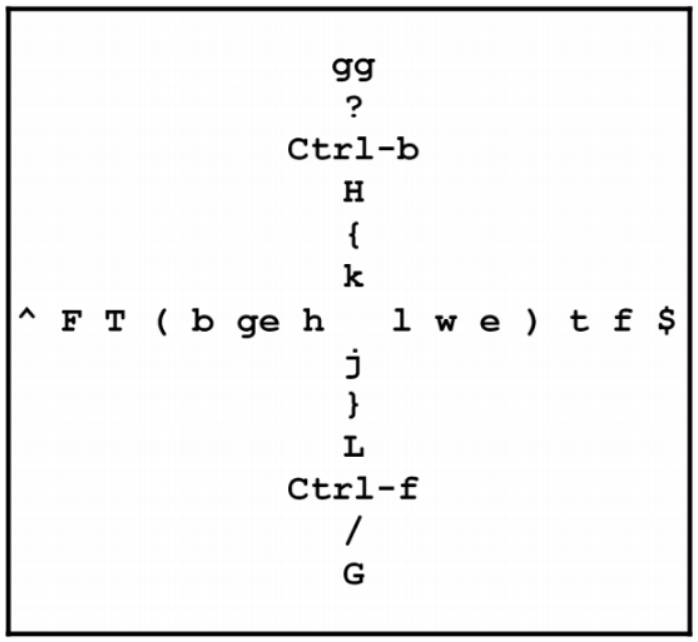
resize +N -> 增加視窗的高度N列

resize -N -> 減少視窗的高度N列

vertical resize +N -> 增加視窗的寬度N列

vertical resize -N -> 減少視窗的寬度N列

* 1. 標籤頁(tab)
     1. 新建標籤頁 -> tabnew or tabnew + file\_name
     2. 依序跳標籤頁 -> gt
     3. 關閉標籤頁 -> tabclose or q
     4. 移動標籤頁在N頁之後 -> tabmove N
     5. 移動標籤頁在0頁的位置 -> tabmove 0
  2. 折疊
     1. 打開或關閉折疊 -> za
  3. 文件瀏覽的滑鼠游標移動方法
     1. 上下左右 -> k, j, h, l
     2. 向前移動word的開頭 -> w, 向後移動word的開頭 -> b
     3. 向前移動word的尾部 -> e, 向後移動word的尾部 -> ge
     4. 滑鼠游標移動到段落開頭, 結尾 -> shift + {, shift + }
     5. 滑鼠游標移動到當前視窗的開頭, 結尾 -> H, L
     6. Pagedown, pageup -> Ctrl + f, Ctrl + b
     7. 滑鼠游標移動到文件的開頭, 結尾 -> gg, G
     8. 滑鼠游標移動到一列內容的頭 -> \_
     9. 滑鼠游標移動到一列內容的尾 -> $
     10. 在同一列向前搜尋字元(結果在字元前) -> t + char
     11. 在同一列向前搜尋字元(結果在字元上) -> f + char
     12. Continue



* 1. 文件瀏覽的搜尋方法
     1. 向前搜尋 -> / + string or char + enter -> 按n向前搜尋
     2. 向後搜尋 -> ? + string or char + enter -> 按n向後搜尋
     3. 取消螢光背景 -> noh
  2. 文件瀏覽跨檔案的搜尋方法 - vimgrep
     1. 搜尋指定string於目錄內檔案 -> vimgrep <string> <path>
     2. <string> -> 欲搜尋的目標
     3. <path> -> \*\*表示在目前目錄內遞迴搜尋
     4. EX:

vimgrep print \*\*/\*.py "search print string within dir & \*.py file.

* + 1. 打開搜尋到的內容視窗 -> copen
    2. 關閉搜尋到的內容視窗 -> q
  1. 進入insert mode滑鼠游標的位置
     1. 滑鼠游標下一個字元位置 -> a
     2. 該列資料的最後位置 -> A
     3. 滑鼠游標目前的位置 -> i
     4. 該列資料的最後位置 -> I
     5. 滑鼠資料列下一列新增列 -> o
     6. 滑鼠資料列上一列新增列 -> O
  2. 刪除列
     1. 刪除一列 -> dd
  3. 修改列(執行完後自動進入insert mode)
     1. 刪除一列後進入insert mode -> cc
     2. 刪除游標字元 -> c + 向右鍵
     3. 刪除游標左邊字元 -> c + 向左鍵
     4. 刪除游標字元 -> s
  4. 刪除(), [], {}括號裡的內容
     1. 刪除()內的內容 -> d + i + (
     2. 刪除[]內的內容 -> d + i + [
     3. 刪除{}內的內容 -> d + i + {
  5. Copy and past
     1. 複製一整列 -> yy
     2. 複製一個單字 -> ye or yw
     3. 貼上 -> p
  6. Buffer

Buffer用數字來當作區分, 每一個buffer代表一個文件.

* + 1. 看有哪些buffer -> ls or buffers or files
    2. 不同buffer之間的轉換 -> b + number, EX: b1
    3. 表示當前視窗對應的buffer符號 -> %
    4. 表示當前buffer為活動狀態的符號 -> a
    5. 表示當前游標所在位置的符號 -> line + number
    6. Close buffer -> bd + number, EX: bd1
    7. 關閉當前視窗的buffer -> bd
  1. Register

Register有多種的種類

* + 1. 看有哪些register -> registers
    2. 看指定register的內容 -> reg + register\_name
    3. 無名register的名稱表示 -> “”
    4. 與p作用相同功能 -> “”p
    5. 一些只讀的register, 其符號所表示的意思:

表示當前文件名的符號 -> %

表示上次打開的文件名 -> #

表示最後在檔案內容所做的事情 -> .

表示最後執行的命令 -> :

* 1. vimdiff比較文件
     1. 兩個文件的比較 -> vimdiff ./a.py ./b.py
     2. 到不同處向前搜尋 -> ]c
     3. 到不同處向後搜尋 -> [c
     4. 修改自己 -> do(or :diffget)
     5. 修改自己全部內容 -> :%diffget
     6. 修改對方 -> dp(or :diffput)
     7. 修改對方全部內容 -> :%diffput
     8. Refresh不同處的標註顯示 -> :diffupdate
  2. Vim顏色目前設定
     1. 顯示顏色設定值 -> highlight
  3. Continue

1. **.vimrc setting**
   1. .vimrc文件的配置可分為以下主題:

“ Edit(編輯) -------------------------------------------------------------------------------------------

“ View(外觀) ------------------------------------------------------------------------------------------

“ Move(移動) -----------------------------------------------------------------------------------------

“ Search(搜索) ----------------------------------------------------------------------------------------

* 1. Continue
  2. 打開自動補全的menu功能

set wildmenu " Enable enhanced tab autocomplete.

" 打開自動補全功能

set wildmode=list:longest,full " Complete till longest string, then open menu.

" 補全允許支援最長字串, 然後打開wildmenu

* + 1. :e -> 空格然後按tab鍵
    2. tab -> 選擇檔案
    3. 上方向鍵 -> 往上層目錄跑, 下方向鍵 -> 往下層方向鍵跑
  1. 視窗間移動按鍵改變對應按鈕

" Navigate windows with <Ctrl-hjkl> instead of <Ctrl-w> followed by hjkl.

noremap <c-h> <c-w><c-h> " 用Ctrl + h, 取代Ctrl + w然後再按h

noremap <c-j> <c-w><c-j> " 用Ctrl + j, 取代Ctrl + w然後再按j

noremap <c-k> <c-w><c-k> " 用Ctrl + k, 取代Ctrl + w然後再按k

noremap <c-l> <c-w><c-l> " 用Ctrl + l, 取代Ctrl + w然後再按l

* 1. 顯示列位置的數字
     1. Command

Number + enter + 前進或後退-> 向number數前進或是後退到指定的位置。

* + 1. setting

set number " 顯示絕對的位置

set relativenumber " 顯示相對的位置

* 1. 顯示搜尋的high light功能

set hlsearch " 顯示搜尋匹配string的螢光背景

set incsearch " 顯示逐步搜尋匹配string的螢光背景

* 1. 設定OS與vim的黏貼系統互通

Set clipboard=unamed,unnamedplus " enable copy text to vim’s \*, + register

* 1. Continue

1. **self package setting**
   1. Netrw
      1. 打開目錄 -> Ex, e ./
      2. 打開上一層目錄 -> e ../
      3. continue
   2. continue
   3. continue
2. **external package setting**
   1. Plug 'scrooloose/nerdtree', { 'on': 'NERDTreeToggle' }
      1. Command

* NERDTree -> 進入啟動套件
* Bookmark -> 指定滑鼠游標對應的目錄為書籤
* B -> 列出所有的書籤
  + 1. Setting
* { 'on': 'NERDTreeToggle' }
* 執行NERDTree才開始啟動套件, 不要拖到vim效能.
* let NERDTreeShowBookmarks = 1 " Display bookmarks on startup.
* 一啟動NERDTree就在視窗上方列出所有被標記為書籤的目錄.
* autocmd VimEnter \* NERDTree " Enable NERDTree on Vim startup.
* 當vim打開的時候, NERDTree就打開.
* autocmd bufenter \* if (winnr("$") == 1 && exists("b:NERDTree") &&

\ b:NERDTree.isTabTree()) | q | endif

* 當最後一個檔案關閉時, NERDTree即自動關閉.
* continue
  1. Plug 'easymotion/vim-easymotion'
     1. Command
* \ + \ + 任何方向鍵 -> 啟動套件
* \ + \ + w -> 向前移動游標位置
* \ + \ + f -> 向前移動指定游標位置
* \ + \ + F -> 向後移動指定游標位置
  + 1. Setting
  1. Plug 'vim-scripts/ScrollColors'
     1. Command
* SCROLL -> 瀏覽不同的配色
* Enter -> 選定顏色
  + 1. Setting
  1. Plug 'flazz/vim-colorschemes'

有大量vim配色的套件.

* + 1. Command
    2. Setting
  1. Plug 'vim-airline/vim-airline', Plug 'vim-airline/vim-airline-themes'

Vim底下的狀態顯示欄位.

* + 1. Command
    2. Setting
  1. Plug 'tpope/vim-fugitive'
     1. Command
        + Gwrite -> **添加**, 同git add.
        + Gcommit -m “title” -m “message” -> **提交**, 同git commit -m “title” -m “message”
        + Gpush -> **推送**, 同git push
        + Gpull --rebase -> 從遠端拉下更新資料, 與local host同步, 同git pull --rebase
        + Gblame -> 看到每一行的提交, 作者, 日期資訊, 同git blame.
          - C -> 看到提交
          - A -> 看到作者
          - D -> 看到日期
        + continue
     2. setting
  2. Continue

1. **vim顏色編輯**
   1. Vim識別三種不同的終端機設定：**term** - 黑白終端； **cterm** - 彩色終端；**gui** - Gvim視窗。
      1. **term**的功能為定義字體的顯示, 可顯示的種類為：bold、underline、reverse、italic或standout。
      2. **cterm**，可以用cterm設定字體為粗體, ctermfg設定前景顏色；ctermbg設定背景顏色。
      3. **gui**，可以用gui設定字體為粗體, guifg設定前景顏色；guibg設定背景顏色。
   2. **設定例子:**
      1. **\*.vim檔案設定:**

**hi LineNr term=none cterm=none ctermfg=15 ctermbg=0 gui=none guifg=#ffffff guibg=#000000**

* + 1. **命令列設定:**

**:highlight Error term=reverse cterm=blod ctermfg=#00afff ctermbg=#a8a8a8**

1. **Continue**
2. **temp**

為了保持各平台下的一致性，我們應該使用[十六進制顏色代碼](http://zh.wikipedia.org/zh/%E7%BD%91%E9%A1%B5%E9%A2%9C%E8%89%B2%E6%A8%A1%E5%BC%8F#.E5.8D.81.E5.85.AD.E8.BF.9B.E5.88.B6.E6.95.B0.E5.AD.97.E8.A1.A8.E7.A4.BA.E6.96.B9.E6.B3.95)來指定顏色。例如使用以下命令，一次定義幾種終端下的配色：