1. [Easymotion](https://github.com/Lokaltog/vim-easymotion" \t "_blank)

let g:EasyMotion\_leader\_key = '<Leader>'

<Leader><Leader> by default

<Leader><Leader>w

<Leader><Leader>e

<Leader><Leader>t

<Leader><Leader>fo

f<char> 可以实现定位到一行中的某个字符，如 fa 会定位到当前光标到行末出现的第一个 “a”。然而如果存在多个 “a”，就得通过 f<number>a 来完成定位。而使用 Easymotion 后，只需要敲击<leader><leader>fa，则该行及该行以下所有的 “a” 都将被高亮并标记序号，此时再按相应的序号即可定位到指定的 “a”。

<leader><leader> 是 Easymotion 默认的引导键，也可以自定义。比如我将其定义为 “f”，这样用 “f” 定位字符只需要按 ff<char> 即可，非常方便。

Easymotion 不仅支持 f，还支持其它定位功能键，如 w、t。不过我 99% 的时间都只用其和 “f” 来配合。

1. Surround（<https://github.com/tpope/vim-surround>）

change

文本 命令 结果

--------------- ------- -----------

"Hello |world!" cs"' 'Hello world!'

"Hello |world!" cs"<q> <q>Hello world!</q>

(123+4|56)/2 cs)] [123+456]/2

(123+4|56)/2 cs)[ [ 123+456 ]/2

<div>foo|</div> cst<p> <p>foo</p>

fo|o! csw' 'foo'!

fo|o! csW' 'foo!'

(| 为光标位置).

adding

Text               Command       New Text  
---------------    -------      -----------  
Hello w|orld!     ysiw)         Hello (world)!  
Hello w|orld!     csw)         Hello (world)!  
fo|o             ysiwt<html>   <html>foo</html>  
foo quu|x baz   yss"         "foo quux baz*"*  
foo quu|x baz   ySS"         "

                                   foo quux baz  
"

Delete

ds

Normal mode

-----------

ds - delete a surrounding

cs - change a surrounding

ys - add a surrounding

yS - add a surrounding and place the surrounded text on a new line + indent it

yss - add a surrounding to the whole line

ySs - add a surrounding to the whole line, place it on a new line + indent it

ySS - same as ySs

Visual mode

-----------

s - in visual mode, add a surrounding

S - in visual mode, add a surrounding but place text on new line + indent it

Insert mode

-----------

<CTRL-s> - in insert mode, add a surrounding

<CTRL-s><CTRL-s> - in insert mode, add a new line + surrounding + indent

<CTRL-g>s - same as <CTRL-s>

<CTRL-g>S - same as <CTRL-s><CTRL-s>

1. Matchit (<https://github.com/vim-scripts/matchit.zip>)

%跳转到下一匹配符号，原有的可以跳转到匹配括号，添加此插件后更强大，可以跳转到匹配的html标签等。

1. Undotree (<https://github.com/mbbill/undotree>)

可视化撤销，可以看到你所做的改变，可视化的恢复到某一状态。

当前位置被标记为>seq<

下一变化被标记为{seq}，可以通过:redo或 <ctrl-r>跳到下一状态

最近的变化标记为[seq]

保存的变化被标记为s，最后被保存的变化被标记为大写S

:UndotreeToggle 打开undo-tree面板，映射nnoremap <F5> :UndotreeToggle<cr>

There are some hotkeys provided by vim to switch between the changes in history, like u, <ctrl-r>, g+, g- as well as the :earlier and :later commands.

在undo-tree面板按？获得快捷键帮助。

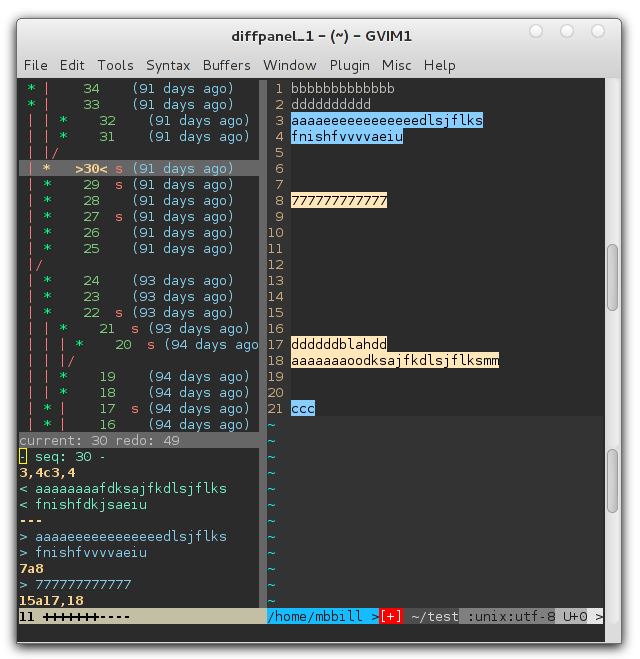
退出后仍保存undo信息

if has('persistent\_undo')

set undofile

set undodir=’~/.undo/’

endif



1. Tagbar(<https://github.com/majutsushi/tagbar>)

比taglist更适合于C++

nmap <silent> <Leader>t :TagbarToggle<cr>

1. nerdtree（<https://github.com/scrooloose/nerdtree>）

比netrw更好用的filesystem explorer，以树形结构显示文件结构，但和winmanager有冲突，无法像netrw一样和谐的整合到winmanager中。按？显示快捷键。

map <C-n> :NERDTreeToggle<CR>

快捷键

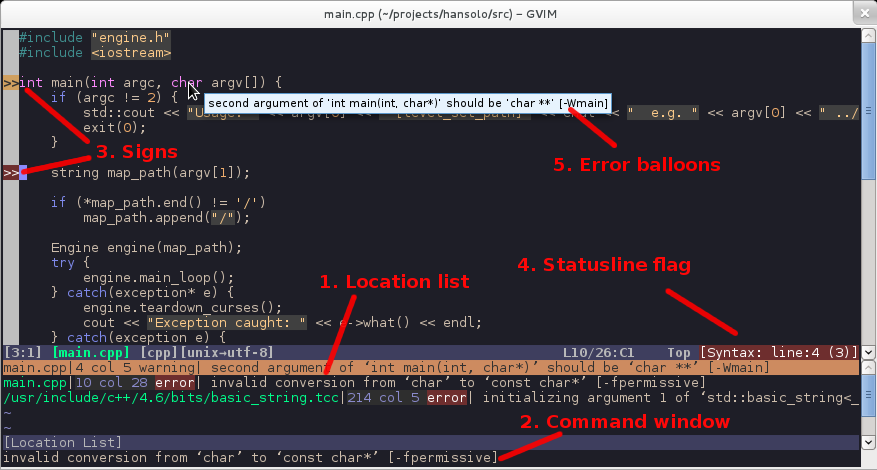
1. Nerdcommenter（<https://github.com/scrooloose/nerdcommenter>）

快速添加注释

1. Ultisnips（<https://github.com/SirVer/ultisnips>）

需要python的支持，很强大的代码补全插件。可以自己定义模板，使用tab键补全。因为与ycm有冲突，重新映射为<C-j>

1. Syntastic(<https://github.com/scrooloose/syntastic>)



let g:syntastic\_che\_on\_open=1

let g:syntastic\_auto\_jump=1

let g:syntastic\_error\_symbol = 'e>'

let g:syntastic\_warning\_symbol = 'w>'

let g:syntastic\_always\_populate\_loc\_list=1

let g:syntastic\_loc\_list\_height = 6

let g:syntastic\_enable\_highlighting = 0

nmap <Leader>e :Errors<cr>

nmap <Leader>c :lnext<cr>

nmap <Leader>z :lpre<cr>

1. singlecompile(<https://github.com/xuhdev/SingleCompile>)

<http://www.topbug.net/SingleCompile/>

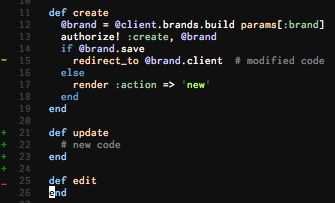
nmap <Leader>g :SCCompileRun<cr><cr><cr>

nmap <Leader>v :SCViewResult<cr>

**:SCCompileRunAsync**

**:SCChooseCompiler**

1. Gitgutter(<https://github.com/airblade/vim-gitgutter>)



let g:gitgutter\_enabled = 1

let g:gitgutter\_highlight\_lines = 0

" let g:gitgutter\_sign\_added = 'xx'

" let g:gitgutter\_sign\_modified = 'yy'

" let g:gitgutter\_sign\_removed = 'zz'

" let g:gitgutter\_sign\_modified\_removed = 'ww'

nmap gh <Plug>GitGutterNextHunk

nmap hg <Plug>GitGutterPrevHunk

You can explicitly turn vim-gitgutter off and on (defaults to on):

* turn off with :GitGutterDisable
* turn on with :GitGutterEnable
* toggle with :GitGutterToggle.

And you can turn line highlighting on and off (defaults to off):

* turn on with :GitGutterLineHighlightsEnable
* turn off with :GitGutterLineHighlightsDisable
* toggle with :GitGutterLineHighlightsToggle.

Furthermore you can jump between hunks:

* jump to next hunk: ]h
* jump to previous hunk: [h.

1. fugitive(<https://github.com/tpope/vim-fugitive>)
2. ctrlp(<https://github.com/kien/ctrlp.vim>)图

let g:ctrlp\_map = '<c-p>'

let g:ctrlp\_cmd = 'CtrlP'

let g:ctrlp\_open\_multiple\_files = 'v' " <C-Z><C-O>时垂直分屏打开多个文件

set wildignore+=\*/tmp/\*,\*.so,\*.swp,\*.zip

let g:ctrlp\_custom\_ignore = {

\ 'dir': '\v[\/]\.(git)$',

\ 'file': '\v\.(log|jpg|png|jpeg)$',

\ }

let g:ctrlp\_working\_path\_mode= 'ra'

let g:ctrlp\_match\_window\_bottom= 1

let g:ctrlp\_max\_height= 10

let g:ctrlp\_match\_window\_reversed=0

let g:ctrlp\_mruf\_max=500

let g:ctrlp\_follow\_symlinks=1

1. unite(<https://github.com/Shougo/unite.vim>)图

可以使用通配符\* |!

如果你在Unite窗口进入插入模式，光标将会移到该窗口的最上方，并显示“>”提示符。输入字符会搜索该列表--这里和FuzzyFinder相似。和常规vim一样，按<ESC>键可以退出插入模式回到命令模式。

命令模式和插入模式都有相应的快捷键映射。比如在命令模式下，当光标在一个文件上，按下a，Unite会显示一个可以操作该文件命令的列表。这个命令列表被称为actions，这个列表可以像Unite其他部分一样被搜索和调用。

actions可以组合通配符。如果输入:Unite file, 然后按下\*将会标记所有文件，再输入a将会列出所有actions，最后选择above，Unite将会打开所有被标记的文件。

nnoremap <leader>f :Unite -start-insert file

1. delimitmate（<https://github.com/Raimondi/delimitMate>）

有争议的插件，我也在试用，自动添加删除匹配的括号、引号