Vim Study

Fan er

目录

前 言

作者简介

- 范儿
- f18811010711@gmail.com
- 个人博客@faner
- 某高校大二自动化专业学生，喜欢Linux系统，日常使用Vim。
- 代码和女朋友比较，女朋友更重要！

本书简介

- Gitbook书写。
- 介绍Vim使用。
- 介绍部分Vim插件。

本书读者

- Vim使用者。
- 需要了解命令行的人。
- Linux使用者及其爱好者。

本书申明

- 非专业书籍，仅供参考！
- 如读者发现笔误，可练习作者，万分感谢。
- 大佬请掠过，本书知识浅薄，不喜勿碰。

写在最后

- 别老看这些，多陪陪家人、女友！

基础操作

Vim基础操作

这些是Vim的基础操作，学会这些操作，你就Vim入门了。

复制粘贴

简单命令

- **y** -- 复o制
- **d** -- 剪切, 和y的用法相同, 使用d替换下面的y就是对应的剪切命令
- **p** -- 粘贴
- **yl** -- 复制当前字符
- **yh** -- 复制当前字符的前一个字符
- **ggyG** -- 全选复制
- **yy** -- 复制当前行
- **nyy** -- 从当前行开始复制n行, 例如5yy就是复制5行
- **yw** -- 复制单词
- **nyw** -- 从光标所在的位置开始往后复制n个单词, 如2yw
- **y$** -- 复制光标的位置到行尾
- **ddp** -- 交换上下两行的位置
- **xp** -- 交换前后两个字符的位置

复杂操作

快速复制文件的某些行

```
bash
// 将9-15的内容复制到16行(以下两种均可), 16行以下的内容会自动下移
:9, 15 copy 16
:9, 15 co 16

// 将9-15 行内容剪切到16行(以下两种均可)
:9, 15 move 16
:9, 15 m 16
```

设置粘贴模式

在粘贴模式下, 粘贴的内容是不会自动缩进的, 因此不会改变粘贴内容的样式,

```
bash
// 进入粘贴模式
:set paste

//退出粘贴模式
:set nopaste
```

在.vimrc中加入下面的代码, 设置切换粘贴"模式的快捷键为

```
bash
set pastetoggle=<F3>
```

与其他文件进行交互

前面加上 `"+` (注意是有加号的)即可, `"` 表示使用寄存器标签, `+` 表示使用系统剪贴板.

```
bash
// 复制5行到系统粘贴板
"+5yy
```

```
// 复制当前行到系统粘贴板
"+yy

// 复制全部内容到系统粘贴板
gg"+yG

// 粘贴当前内容
"+p
```

通过上面的操作我们可以将内容复制到系统剪贴板, 也可以从系统剪贴板中获得数据.
在Linux系统中除了系统粘贴板, 还有一个叫做选择缓冲区, 它和系统剪贴板不同的是选择缓冲区时是实时的, 当我们选择了内容之后, 内容就已经存在选择缓冲区里了, 这意味着如果我们使用选择缓冲区就不用再ctrl+c复制了, 只需用鼠标选择, 有了内容之后下一步就是粘贴了, 使用 `Shift` + `Insert` 或者鼠标中键就可以粘贴选择缓冲区的内容, 在vim中使用两种方式都可以, 但是在gedit等gui编辑器中, 只能通过鼠标中键进行粘贴.
下面是具体流程:

1. 选择, 在vim中下面两种方式均可, 在其他软件中貌似只可以用鼠标选择
    - 在一般模式下按 `v` 或者 `Ctrl` + `v` , 进入可视模式, 此时就可以选择内容了, 选择完后按ESC退出可视模式
    - 使用鼠标直接选择(我试的是直接选择就行,不需要配置, 如果直接选择不行, 在.vimrc中加上 `set mouse=v`)
2. 粘贴
    - `Shift` + `Insert` (编辑模式下)
    - 鼠标中键
    - `"*p` (一般模式下)

如果要更加方便, 可以修改vim默认使用的寄存器改为选择缓冲区, 在.vimrc文件中加入下面的内容:

```
set clipboard = unnamed
```

这样就可以直接使用y, p和系统选择缓冲区进行数据交换了. 即选择完后直接按p即可

华丽的分割线, 下面补充几点:

1. 如果设置y和p默认使用系统剪贴板, 则加入

    ```
    set clipboard=unnamedplus
    ```

2. 在命令模式下输入 `reg` 可以查看寄存器, 如果没有 `*` 和 `+` 寄存器, 输入下面命令安装

    ```
    sudo apt-get install vim-gnome
    ```

3. 使用示例

    ```
    // 全选
    ggyG
    // 粘贴到其他地方
    鼠标中键
    ```

4. 本节参考自 Vim复制粘贴探秘

删除

在vim中, 删除很多时候其实就是将当前内容剪切了(只剪切不粘贴), 使用d操作符, 下面如果不特殊说明就是在一般模式下

简单操作

- **dd** -- 删除当前行
- **ndd** -- 从当前行开始往下删除n行
- **:n,md** -- 删除n到m行, 如 `:1,10d` 删除1到10行的内容,注意是在底行模式下哦
- **d0** -- 删除光标所在位置至行首的字符
- **d$** -- 删除光标所在位置至行尾的字符
- **dw** -- 删除光标之后单词的剩余部分
- **db** -- 删除光标之前单词的剩余部分(如果光标在当前单词的开头, 则删除上一个单词)
- **diw** -- 删除光标上的单词 (不包括空白字符)
- **daw** -- 删除光标上的单词 (包括空白字符)
- **dgg** -- 删除光标所在行至文件开头的内容, 包含光标所在行
- **dG** -- 删除光标所在行至文件结尾的内容, 包含当前行
- **x(**小写**)** -- 删除光标所在的字符
- **X(**大写**)** -- 删除前一个字符
- **Ctrl+u(**编辑模式下**)** -- 删除光标所在位置至行首的字符, 挺有用的, `End Ctrl+u` 就可以在编辑模式下删除整行.

复杂操作

快速删除某段内容

比如有段文字如下:

```
11111rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
nnnnnnnnnnnnnnnnnnnnnnnnnnnnn
fffffffffffffffff
ggggggggggggggggggg33333
```

想要删除11111和33333之间的内容, 可以将文件改成下面的内容

```
11111{rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
nnnnnnnnnnnnnnnnnnnnnnnnnnnnn
fffffffffffffffff
ggggggggggggggggggg}33333
```

即在11111后面加上个 `{` , 在33333前面加上个 `}` , 然后将光标移到 `{` 之前的1上(也就是最后一个1), 输入 `da{` 就可以把11111和33333之间的内容删除了.
另外，除了可以使用{ 外，还可以使用 `"`, `'`, `(` 等成对的特殊符号。

参考自: vim 快速删除指定的一段字符

选择

选择**10-20**行的内容

```
// 底行模式下输入10，跳转到第10行
:10

// 一般模式下输入下面命令，注意是大写V
V20G
```

移动

以下如果不特殊声明就是在一般模式下

```
// 跳转到行首
0

// 跳转到本行第一个不是空白符的地方(空白符就是空格,tab, 换行, 回车等)
^

// 跳转到行尾
$

// 跳转到文件开头
gg

// 跳转到文件结尾
G

// 跳转到第n行行首
nG 或者 ngg

// 底行模式下输入n, 回车跳转到第n行, 如:10 跳转到第10行
:n

// 跳转到本行中光标右侧 {char} 首次出现的位置, {char}表示一个字符,比如fa 跳转到光标右侧a字符首次出现的位置
f{char}

// 跳转到本行跳转到本行中光标左侧 {char} 首次出现的位置
F{char}

// 跳转到本行中光标右侧 {char} 首次出现位置的前一个字符
t{char}

//跳转到本行中光标左侧 {char} 首次出现位置的后一个字符
T{char}

// 到下一个单词的开头
w (默认形式)  W(空白符作为单词分割符)

// 到当前单词的开头
b (默认形式)  B(以空白符为单词分割符)

// 到下一个单词结尾
e (默认形式)  E(以空白符作为单词分割符)

// 括号匹配, 包括(,[,{, 需要先将光标移到括号上,然后会跳转到对应的那个括号
%

// 匹配光标所在的单词, 移动光标到下一个或者上一个匹配单词
* (移动到下一个匹配单词)   #(移动到上一个匹配单词)

//搜索pattern的字符串, 如果出现多个,可按n键跳转到下一个, 如/abc 搜索abc出现的地方
/pattern (往下找)  ?pattern(往上找)
/\<pattern\> 完全匹配

// 移动光标到屏幕上方
H

// 移动光标到屏幕中间
M

// 移动光标到屏幕下方
L

// 设置书签
ma (在当前光标记一个书签, 名字为a, 书签名必须为小写字母, 对用户不可见)
`a (跳转到书签a处, `不是单引号, 而是按键1左边的那个按键)

/**
 * 下面两条指令的跳转的规则: 必须是执行了下面中的一条指令
 * A "jump" is one of the following commands: "'", "`", "G", "/", "?", "n",
 *    "N", "%", "(", ")", "[[", "]]", "{", "}", ":s", ":tag", "L", "M", "H"
 */

// 回到上一次编辑的地方
```

```
ctrl+o

// 回到下一次编辑的地方
ctrl+i
```

块编辑

选择文本块

1. 移动光标到要编辑的位置
2. Ctrl+v 进入快选择模式
3. 移动光标选择同时编辑的行

操作(选择完文本块后)

```
// 修改字符
1. 按`r`(小写), 进入修改模式
2. 输入字符(只能输入一个, 选择的内容都会改为这个字符)

// 在块前面插入内容
1. 按`I`(大写), 进入行首插入模式
2. 输完内容按ESC

// 在块后面插入内容
1. 按`A`(大写), 进入行尾插入模式
2. 输完内容按ESC

// 删除选择内容
d
```

替换

```
当前行进行替换
:s/abc/efg/
:s/abc/efg/g

所有行进行替换
:%s/abc/efg/
:%s/abc/efg/g

从第n行开始向下的所有行进行替换，当n为"."时，表示从当前行开始
:n,$s/abc/efg/
:n,$s/abc/efg/g

只匹配整个单词word，类似于aword不匹配，即在单词前后加上 \< 和 \>
:%s/\<word\>/newword/g

\c忽略abc的大小写，\C不忽略大小写
: %s/abc\c/efg/g

后面的c表示每次替换之前都需要确认
: %s/abc/efg/gc
```

上面命令中，最后没有g表示只替换一行中第一次出现的字符串abc为efg。而后面带g的表示当前行的所有abc替换efg。

执行**shell**命令

执行单条**shell**命令

```
// 在命令前面加个!号, 执行万命令后会提示返回vim
:! {command}

// 执行命令,并读取其输出(是将命令的输出写入到当前的打开文件中)
:r !{command}

// 执行命令, 并将命令的输出传送至vim命令模式输入(即在命令行那里输出)
:w !{command}
```

打开**shell**窗口

```
//在vim中输入,就会挂起vim, 进入到shell执行框
:shell

//执行完shell命令,使用exit返回到vim
exit
```

分屏

打开文件时分屏

```
vim -On file1 file2 ...
vim -on file1 file2 ...
```

其中大写 `O` 表示垂直分割(vertical), 小写 `o` 表示水平分割(horizontal, 默认值), `n` 表示分屏数量, 如果不加文件名, 则打开 `n` 个空白分屏, 如果不加 `n` , 则根据打开的文件数量来决定分屏的数量

在**vim**中分屏

```
// 创建空白分屏, 垂直分割
:vsp

// 打开新文件, 垂直分割
:vsp filename

// 创建空白分屏, 水平分割
:sp 或者
:new

// 打开新文件, 水平分割
:sp filename 或者
:new filename

// 快捷键,
Ctrl + w + s 水平分割
Ctrl + w + v 垂直分割
```

关闭分屏

```
// 关闭其他分屏, 只保留当前分屏
Ctrl + w + o 或者
:only

// 关闭当前分屏, 不能关闭最后一个窗口
Ctrl + w + c

// 关闭当前分屏, 只剩最后一个分屏时退出
Ctrl + w + q
```

分屏之间移动

```
// 分屏之间移动光标
Ctrl + w + w

// 在分屏之间移动光标
上 - Ctrl + w + k 或者 Ctrl + 上箭头
下 - Ctrl + w + j 或者 Ctrl + 下箭头
左 - Ctrl + w + h 或者 Ctrl + 左箭头
右 - Ctrl + w + l 或者 Ctrl + 右箭头

// 移动分屏(不是移动光标), 先按Ctrl + w, 再按shift + hjkl
上 - Ctrl + w + K
下 - Ctrl + w + J
左 - Ctrl + w + H
右 - Ctrl + w + L
```

改变尺寸

```
// 先按Ctrl + w, 再按后面的字符
左 - Ctrl + w + <
右 - Ctrl + w + >
上 - Ctrl + w + +  // 最后一个加号是按键
下 - Ctrl + w + -
均等 - Ctrl + w + =
```

处理二进制文件

- 打开文件时需要加上 `-b` 选项, 否则会在文件后面加上**0x0a**

```
vim -b filename
```

- 在命令模式下输入面的命令, 使用**xxd**转换成十六进制格式

```
:%!xxd
```

其中 `%` 表示当前文件的路径, `!` 表示执行一次shell命令, 然后就可以以16进制的格式修改文件

- 修改完成后,需要执行下面的命令将十六进制转为二进制再保存

```
:%!xxd -r
:w
```

---

vim 编辑二进制实际上使用的是xxd, 下面是xxd的一些选项

- -b : 以二进制的格式打开文件, 在vim中不要使用这个选项, 因为使用-r还原为二进制的时候还是按十六进制还原的
- -g : 输出时每组的8位字节数, 默认是2

```
// -g 0
0000000: 00000000010000000200000003000000  ................
// -g 1
0000000: 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00  ................
// -g 2
0000000: 0000 0000 0100 0000 0200 0000 0300 0000  ................
```

- -r : 将十六进制转换为二进制

大小写转换

~	将光标下的字母改变大小写
3~	将光标位置开始的3个字母改变其大小写
g~~	改变当前行字母的大小写
U	将可视模式下选择的字母全改成大写字母
u	将可视模式下选择的字母全改成小写
gUU	将当前行的字母改成大写
3gUU	将从光标开始到下面3行字母改成大写
guu	将当前行的字母全改成小写
gUw	将光标下的单词改成大写。
guw	将光标下的单词改成小写。
ggguG/gggUG	将整篇文章变为小写/大写
:h ~	查看更多使用方法

原文地址

输入特殊字符

方法

在输入模式下按 `Ctrl + k`，然后输入特殊字符对应的 `digraph` 值（二和字符）即可。
例如输入 Ω，可以先按 `Ctrl + k`，然后输入 `W*` (W是大写）。
在底行模式下输入 `:dig` 就可以查看 特殊字符和 `digraph` 的对应表。

常用字符

- **Δ -** `D*`
- **Ω -** `W*`
- **Θ -** `H*`

Vimrc 配置

设置折叠

配置

```
" 基于缩进
"set foldmethod=indent
" 基于语法
set foldmethod=syntax
" 启动vim时关闭折叠代码
set nofoldernable
" 使用空格来开关折叠
nnoremap <space> @=((foldclosed(line('.')) < 0) ? 'zc' : 'zo')<CR>
```

配置快捷键

插入模式下移动使用 ctrl + h/j/k/l 移动光标

```
inoremap jk <esc>
:imap <C-h> <Left>
:imap <C-j> <Down>
:imap <C-k> <Up>
:imap <C-l> <Right>
:imap <C-o> <End>
```

快捷键

- `zc`：关闭折叠
- `zo`：打开折叠
- `za`：打开或者关闭当前折叠
- `zM`：关闭所有折叠
- `zR`：打开所有折叠

正常模式下插入空格

配置

```
nmap <silent> to :call append('.', '')<CR>j
nmap <silent> tO :call append(line('.')-1, '')<CR>k
```

快捷键

使用 `to` 或者 `tO` 组合键

头文件和源文件互换

```
nnoremap <silent> <F12> :A<CR>
```

Vimrc 文件

```
set nocompatible              " be iMproved, required
filetype off                  " required

" set the runtime path to include Vundle and initialize
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()
" alternatively, pass a path where Vundle should install plugins
"call vundle#begin('~/some/path/here')

" let Vundle manage Vundle, required
Plugin 'VundleVim/Vundle.vim'
Plugin 'Valloric/YouCompleteMe'
"自动补全
Bundle 'Raimondi/delimitMate'
filetype plugin indent on

" The following are examples of different formats supported.
" Keep Plugin commands between vundle#begin/end.
" plugin on GitHub repo
"Plugin 'tpope/vim-fugitive'
" plugin from http://vim-scripts.org/vim/scripts.html
"Plugin 'L9'
" Git plugin not hosted on GitHub
"Plugin 'git://git.wincent.com/command-t.git'
" git repos on your local machine (i.e. when working on your own plugin)
"Plugin 'file:///home/gmarik/path/to/plugin'
" The sparkup vim script is in a subdirectory of this repo called vim.
" Pass the path to set the runtimepath properly.
"Plugin 'rstacruz/sparkup', {'rtp': 'vim/'}
" Avoid a name conflict with L9
"Plugin 'user/L9', {'name': 'newL9'}

" All of your Plugins must be added before the following line
call vundle#end()            " required
filetype plugin indent on    " required
" To ignore plugin indent changes, instead use:
"filetype plugin on
"
" Brief help
" :PluginList       - lists configured plugins
" :PluginInstall    - installs plugins; append `!` to update or just :PluginUpdate
" :PluginSearch foo - searches for foo; append `!` to refresh local cache
" :PluginClean      - confirms removal of unused plugins; append `!` to auto-approve removal
"
" see :h vundle for more details or wiki for FAQ
" Put your non-Plugin stuff after this line

"markdown
Plugin 'godlygeek/tabular'
Plugin 'plasticboy/vim-markdown'
Plugin 'terryma/vim-multiple-cursors'


set number                  " 显示行号
set cursorline              " 突出显示当前行


"set guicursor=i:block-iCursor-blinkon0,v:block-vCursor
"set cursorcolumn             " 高亮显示当前行/列
set shiftwidth=4            " 设定 << 和 >> 命令移动时的宽度为 4
set softtabstop=4           " 使得按退格键时可以一次删掉 4 个空格
set tabstop=4               " 设定 tab 长度为 4
set expandtab               " 加上这句防止代码复制到别的地时tab变为8个空格
set autochdir               " 自动切换当前目录为当前文件所在的目录

"color solarized
set ruler                     " 显示光标当前位置
set nowrap                    " 禁止折行
syntax enable                 " 开启语法高亮功能
syntax on                     " 允许用指定语法高亮配色方案替换默认方案
"colorscheme  anderson  " 设定配色方案
"colorscheme  spring     " 设定配色方案
"color spring
set smartindent             " 开启新行时使用智能自动缩进
```

```
set laststatus=2            " 显示状态栏 (默认值为 1, 无法显示状态栏)
"set foldenable             " 开始折叠
"set foldmethod=syntax      " 设置语法折叠

set mouse=a

"设置补全文件名
set wildmode=list:longest

"hi preproc        guifg=blue        ctermfg=blue


" MiniBufExplorer     多个文件切换 可使用鼠标双击相应文件名进行切换
let g:miniBufExplMapWindowNavVim=1
let g:miniBufExplMapWindowNavArrows=1
let g:miniBufExplMapCTabSwitchBufs=1
let g:miniBufExplModSelTarget=1

map <F7> 20zl " Scroll 20 characters to the right
map <F8> 20zh " Scroll 20 characters to the left

"au VimEnter * if line('$') > &lines | set go+=r | else | set go-=r | endif
"au VimResized * if line('$') > &lines | set go+=r | else | set go-=r | endif

" 配置多语言环境
if has("multi_byte")
    " UTF-8 编码
    set encoding=utf-8
    set termencoding=utf-8
    set formatoptions+=mM
    set fencs=utf-8,gbk
    if v:lang =~? '^\(zh\)\|\(ja\)\|\(ko\)'
        set ambiwidth=double
    endif
    if has("win32")
        source $VIMRUNTIME/delmenu.vim
        source $VIMRUNTIME/menu.vim
        language messages zh_CN.utf-8
    endif
else
    echoerr "Sorry, this version of (g)vim was not compiled with +multi_byte"
endif

"配置taglist
let Tlist_Show_One_File=1   "只显示一个文件中的tag
let Tlist_Exit_OnlyWindow=1
let Tlist_Sort_Type="name" "按tag名称排序
let Tlist_Use_SingleClick=1  "单击
let Tlist_Use_Right_Window=1 "把taglist窗口放在屏幕的右侧，缺省在左侧
let Tlist_Show_Menu=1 "显示taglist菜单
map <silent> <F9> :TlistToggle<cr>


"配置ctags快捷键
map <C-F12> :!ctags --c-kinds=+lpx -R .<CR>

"配置WinManager
let g:winManagerWindowLayout='FileExplorer'
nmap wm :WMToggle<cr>

"设置cscope
:set cscopequickfix=s-,c-,d-,i-,t-,e-

"设置MiniBufExplorer
let g:miniBufExplMapWindowNavArrows = 1

"头文件互换
nnoremap <silent> <F12> :A<CR>

let g:ycm_global_ycm_extra_conf = '/home/zhangjikai/.vim/bundle/third_party/ycmd/cpp/ycm/ycm.c.py'
let g:ycm_semantic_triggers = {}
let g:ycm_semantic_triggers.c = ['->', '.', '(', '[', '&']
let g:ycm_collect_identifiers_from_tags_files=1
let g:ycm_min_num_of_chars_for_completion=3
let g:ycm_seed_identifiers_with_syntax=1


" YouCompleteMe 功能
" 补全功能在注释中同样有效
let g:ycm_complete_in_comments=0
" 允许 vim 加载 .ycm_extra_conf.py 文件，不再提示
let g:ycm_confirm_extra_conf=0
" 开启 YCM 基于标签引擎
```

```
let g:ycm_collect_identifiers_from_tags_files=1
" 引入 C++ 标准库tags，这个没有也没关系，只要.ycm_extra_conf.py文件中指定了正确的标准库路径
set tags+=/data/misc/software/misc./vim/stdcpp.tags
" YCM 集成 OmniCppComplete 补全引擎，设置其快捷键
inoremap <leader>; <C-x><C-o>
" 补全内容不以分割子窗口形式出现，只显示补全列表
set completeopt-=preview
" 从第一个键入字符就开始罗列匹配项
let g:ycm_min_num_of_chars_for_completion=1
" 禁止缓存匹配项，每次都重新生成匹配项
let g:ycm_cache_omnifunc=0
" 语法关键字补全
let g:ycm_seed_identifiers_with_syntax=1
" 修改对C函数的补全快捷键，默认是CTRL + space，修改为ALT + ;
let g:ycm_key_invoke_completion = '<M-;>'
" 设置转到定义处的快捷键为ALT + G，这个功能非常赞
nmap <M-g> :YcmCompleter GoToDefinitionElseDeclaration <C-R>=expand("<cword>")<CR><CR>
" 设置按哪个键上屏
let g:ycm_key_list_select_completion = ['<TAB>', '<Down>', '<Enter>']

"括号补全


"取消自动检查错误

let g:syntastic_c_checkers = []
let g:syntastic_disabled_filetypes = ['c']
let g:syntastic_ignore_files = ['*.h','*.c']

"自动保存
let g:auto_save = 1
let g:auto_save_no_updatetime = 0
let g:auto_save_in_insert_mode = 0

"配置gvim
if has("gui_running")
set guifont=YaHei\ Consolas\ Hybrid\ 12
endif


" Go to last file(s) if invoked without arguments.
"autocmd VimLeave * nested if (!isdirectory($HOME . "/.vim")) |
"   \ call mkdir($HOME . "/.vim") |
"   \ endif |
"   \ execute "mksession! " . $HOME . "/.vim/Session.vim"

"autocmd VimEnter * nested if argc() == 0 && filereadable($HOME . "/.vim/Session.vim") |
"   \ execute "source " . $HOME . "/.vim/Session.vim"

"设置terminal 使用gui颜色
" IMPORTANT: Uncomment one of the following lines to force
" using 256 colors (or 88 colors) if your terminal supports it,
" but does not automatically use 256 colors by default.
set t_Co=256
"set t_Co=88
let g:CSApprox_attr_map = { 'bold' : 'bold', 'italic' : '', 'sp' : '' }
colorscheme spring
hi CursorLine   cterm=NONE ctermbg=lightgray ctermfg=black  guifg=white

au BufRead,BufNewFile *.{md,mdown,mkd,mkdn,markdown,mdwn} set filetype=markdown nofoldenable

"设置粘贴模式切换,在粘贴模式下,粘贴时不会自动缩进
set pastetoggle=<F3>
set clipboard=unnamed

"正常模式下插入空行 to/tO
nmap <silent> to :call append('.', '')<CR>j
nmap <silent> tO :call append(line('.')-1, '')<CR>k

set foldmethod=syntax
set nofoldenable
```

插件简介

Vim作为一款神级编辑器，有很多人为他编写了许多的插件，其中有很多非常优秀的插件。这列举了一些笔者经常使用的插件，欢迎读者补充扩展。

Vundle

安装

vundle是vim的一个插件管理器,使用它可以方便的安装管理vim的各种插件,下面是它的github地址,下面是安装流程(具体以github上的说明为主):

- 下载vundle

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

- 配置文件
  将下面配置copy到~/.vimrc文件中

```
set nocompatible              " be iMproved, required
filetype off                  " required

" set the runtime path to include Vundle and initialize
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()
" alternatively, pass a path where Vundle should install plugins
"call vundle#begin('~/some/path/here')

" let Vundle manage Vundle, required
Plugin 'VundleVim/Vundle.vim'

" The following are examples of different formats supported.
" Keep Plugin commands between vundle#begin/end.
" plugin on GitHub repo
Plugin 'tpope/vim-fugitive'
" plugin from http://vim-scripts.org/vim/scripts.html
"Plugin 'L9'
" Git plugin not hosted on GitHub
Plugin 'git://git.wincent.com/command-t.git'
" git repos on your local machine (i.e. when working on your own plugin)
"Plugin 'file:///home/gmarik/path/to/plugin'
" The sparkup vim script is in a subdirectory of this repo called vim.
" Pass the path to set the runtimepath properly.
Plugin 'rstacruz/sparkup', {'rtp': 'vim/'}
" Avoid a name conflict with L9
"Plugin 'user/L9', {'name': 'newL9'}

" All of your Plugins must be added before the following line
call vundle#end()            " required
filetype plugin indent on    " required
" To ignore plugin indent changes, instead use:
"filetype plugin on
"
" Brief help
" :PluginList       - lists configured plugins
" :PluginInstall    - installs plugins; append `!` to update or just :PluginUpdate
" :PluginSearch foo - searches for foo; append `!` to refresh local cache
" :PluginClean      - confirms removal of unused plugins; append `!` to auto-approve removal
"
" see :h vundle for more details or wiki for FAQ
" Put your non-Plugin stuff after this line
```

- 安装插件
  打开vim, 输入:PluginInstall

命令

- **:BundleList** -列举出列表中(.vimrc中)配置的所有插件
- **:BundleInstall** -安装列表中全部插件
- **:BundleInstall!** -更新列表中全部插件

- **:BundleSearch foo -**查找foo插件
- **:BundleSearch! foo -**刷新foo插件缓存
- **:BundleClean -**清除列表中没有的插件
- **:BundleClean! -**清除列表中没有的插件

YouCompleteMe

地址

安装

Vundle

```
Plugin 'Valloric/YouCompleteMe'
```

Cmake

```
sudo apt-get install build-essential cmake
```

c语言支持安装

```
cd ~/.vim/bundle/YouCompleteMe
./install.py --clang-completer
```

配置

vimrc配置

```
let g:ycm_global_ycm_extra_conf = '/home/zhangjikai/.vim/bundle/YouCompleteMe/third_party/ycmd/cpp/ycm/.ycm_e
xtra_conf.py'
let g:ycm_semantic_triggers = {}
let g:ycm_semantic_triggers.c = ['->', '.', '(', '[', '&']
let g:ycm_collect_identifiers_from_tags_files=1
let g:ycm_min_num_of_chars_for_completion=3
let g:ycm_seed_identifiers_with_syntax=1

" YouCompleteMe 功能
" 补全功能在注释中同样有效
let g:ycm_complete_in_comments=0
" 允许 vim 加载 .ycm_extra_conf.py 文件，不再提示
let g:ycm_confirm_extra_conf=0
" 开启 YCM 基于标签引擎
let g:ycm_collect_identifiers_from_tags_files=1
" 引入 C++ 标准库tags，这个没有也没关系，只要.ycm_extra_conf.py文件中指定了正确的标准库路径
set tags+=/data/misc/software/misc./vim/stdcpp.tags
" YCM 集成 OmniCppComplete 补全引擎，设置其快捷键
inoremap <leader>; <C-x><C-o>
" 补全内容不以分割子窗口形式出现，只显示补全列表
set completeopt-=preview
" 从第一个键入字符就开始罗列匹配项
let g:ycm_min_num_of_chars_for_completion=1
" 禁止缓存匹配项，每次都重新生成匹配项
let g:ycm_cache_omnifunc=0
" 语法关键字补全
let g:ycm_seed_identifiers_with_syntax=1
" 修改对C函数的补全快捷键，默认是CTRL + space，修改为ALT + ;
let g:ycm_key_invoke_completion = '<M-;>'
" 设置转到定义处的快捷键为ALT + G，这个功能非常赞
nmap <M-g> :YcmCompleter GoToDefinitionElseDeclaration <C-R>=expand("<cword>")<CR><CR>
" 设置按哪个键上屏
let g:ycm_key_list_select_completion = ['<TAB>', '<Down>', '<Enter>']
```

ycm_extra_conf.py 配置

支持提示头文件

在 `flag=[*]` 里加入

```
  '-isystem',
  '/usr/include'
```

markdown

安装

在 `~/.vimrc` 中加入:

```
Plugin 'godlygeek/tabular'
Plugin 'plasticboy/vim-markdown'
```

github地址

配置

```
" 设置使用markdown插件的类型以及不自动折叠代码
au BufRead,BufNewFile *.{md,mdown,mkd,mkdn,markdown,mdwn} set filetype=markdown nofoldenable
```

MiniBufExplorer

安装

```
Bundle 'fholgado/minibufexpl.vim'
```

github地址

快捷键

- **Tab** : 向前循环切换到每个buffer上
- **Shift - Tab** : 向后循环切换到每个buffer上
- **Enter** : 打开光标所在的buffer
- **d** : 删除光标所在的buffer

命令

在一般模式下

```
// 打开当前buffer的下一个buffer
:bn

// 打开当强buffer的上一个buffer
:bp

// 打开编号为<num>(即每个buffer前面的数字)的buffer,
:b<num>
```

Taglist

安装

```
Bundle 'taglist.vim'
```

github地址

配置

```
"只显示一个文件中的tag
let Tlist_Show_One_File=1

如果 taglist 窗口是最后一个窗口，则退出 vim
let Tlist_Exit_OnlyWindow=1

"按tag名称排序
let Tlist_Sort_Type="name"

"鼠标单击跳转到tag定义，要开启鼠标功能
let Tlist_Use_SingleClick=1

"把taglist窗口放在屏幕的右侧，缺省在左侧
let Tlist_Use_Right_Window=1

"显示taglist菜单
let Tlist_Show_Menu=1

"映射taglist打开关闭的快捷键
map <silent> <F9> :TlistToggle<cr>
```

使用

```
// 跳转到tag定义处
<enter>

// 在一个新窗口中跳转到tag的定义
o

// 预览tag的定义(光标还在taglist窗口内)
p

// 显示tag的原型(在vim窗口的最下方，底行模式的地方)
<space> 空格

// 更新taglist列表
u

// 更改排序方式，在按名字排序和按出现顺序排序间切换
s

// 删除taglist列表
d

// 放大和缩小taglist窗口
x

// 展开一个折叠
+

// 将tag折叠起来
-

// 展开所有的折叠
*

// 将所有的tag折叠起来
```

```
=

// 移动到前一个文件的taglist(如果允许显示多个文件的tag)
[[

// 移动到后一个文件的taglist
]]

// 关闭taglist窗口
q

// 打开/关闭帮助
<F1>
```

ctags

安装

```
sudo apt-get install ctags
```

配置

```
" 配置Ctrl + F12为生产ctags的快捷键
map <C-F12> :!ctags -R .<CR>
```

使用

```
// 跳转到变量或函数定义处
Ctrl + ]

// 返回上一个地方
Ctrl + t

// 跳转到局部变量的定义处(但是好像在不同的函数内有相当的局部变量声明时，都会跳到第一个)
gd 或者 [ + <Tab>
```

查询

```
// 查看可以支持的语法元素
ctags --list-kinds

// 查看支持的c语法元素
ctags --list-kinds=c

// 查看ctags支持的语言
ctags --list-languages

// 查看每种语言对应的扩展名
ctags --list-maps

// 使生成的标签支持支持局部变量，但是局部变量只能跳转到第一个声明的地方
ctags --c-kinds=+lpx -R .
```

NERDTree

树型文件列表

安装

地址

```
Bundle 'scrooloose/nerdtree'
```

配置

```
let NERDTreeWinPos='right'
let NERDTreeWinSize=30
" 命令是:NERDTree
map <F2> :NERDTreeToggle<CR>
```

WinManager

窗口管理器

安装

```
Plugin 'vim-scripts/winmanager'
```

配置

```
let g:winManagerWindowLayout='FileExplorer'
nmap wm :WMToggle<cr>


" 配置Ctrl + 方向键在窗口之间切换
nmap <silent> <C-Up> :wincmd k<CR>
nmap <silent> <C-Down> :wincmd j<CR>
nmap <silent> <C-Left> :wincmd h<CR>
nmap <silent> <C-Right> :wincmd l<CR>
```

Vim-multiple-cursors(vim多重光标选取插件)

安装

使用vundle安装, 在.vimrc中加入

```
Plugin 'terryma/vim-multiple-cursors'
```

这里是github地址

使用

上图的按键顺序: `2Gfp<C-n><C-n><C-n>cname`

1. `2G` - 跳转到第2行
2. `fp` - 移动光标到字符p
3. `<C-n>` - 即 `Ctrl+n` , 选择当前单词 `poorly_named_var`
4. `<C-n><C-n>` - 每按一次 `<C-n>` 向下选取一个同样的单词, 这里选择下面的两个 `pooly_named_var`
5. `c` - 按c开始修改
6. `name` - 输入要修改的内容, 这里是 `name`
7. 按 `<Esc>` 回到正常的模式

其他插件

```
Bundle 'minibufexpl.vim' //buffer管理
Bundle 'comments.vim' //快速注释
Bundle 'winmanager' //窗口管理

Bundle 'Valloric/YouCompleteMe' //强大的自动补全，谁用谁知道
Bundle 'Lokaltog/vim-powerline' //漂亮的状态栏
Bundle 'kien/ctrlp.vim' //强大的文件搜索
Bundle 'godlygeek/tabular' //快速对齐
Bundle 'terryma/vim-multiple-cursors' //多光标同时编辑
Bundle 'tpope/vim-haml' //sass scss haml等css开发语言支持
Bundle 'genoma/vim-less' //less支持
Bundle 'Raimondi/delimitMate' //自动补全引号 括号等
Bundle 'hail2u/vim-css3-syntax' //css3语法高亮
Bundle 'othree/html5.vim' //html5标签支持
Bundle 'docunext/closetag.vim' //html xml自动闭合标签
Bundle 'gregsexton/MatchTag' //自动高亮匹配标签
Bundle 'easymotion/vim-easymotion' //强大的搜索定位
Bundle 'terryma/vim-expand-region' //自动选择括号等符号中的内容
Bundle 'tpope/vim-surround' //符号自动环绕
Bundle 'tpope/vim-repeat' //更为强大的重做功能
Bundle 'bronson/vim-trailing-whitespace' //显示以及去除行尾空格
Bundle 'SirVer/ultisnips' //代码片段补全
Bundle 'honza/vim-snippets' //内置了一堆语言的自动补全片段
Bundle 'rstacruz/vim-ultisnips-css' //css的补全
Bundle 'tacahiroy/ctrlp-funky' //基于ctrlp的搜索函数等变量名
Bundle 'dyng/ctrlsf.vim' //基于ctrlp的文件内容搜索，配合vim－multiple－cursors可以很方便一次修改多个文件的内容
Bundle "pangloss/vim-javascript" //更好的js语法 锁进支持
Bundle 'othree/yajs.vim' //更好的js语法高亮
Bundle 'othree/javascript-libraries-syntax.vim' //js各类框架 库的高亮支持
Bundle 'maksimr/vim-jsbeautify' //格式化js css等
Bundle 'marijnh/tern_for_vim' //牛逼的基于语法分析的补全
Bundle 'mbriggs/mark.vim' //标记高亮
Bundle 'gorodinskiy/vim-coloresque' //颜色符号显示对应颜色
Bundle 'scrooloose/nerdtree' //文件树浏览
Bundle 'Xuyuanp/nerdtree-git-plugin' //git支持
Bundle 'majutsushi/tagbar' //ctags标签提取显示
```