The missing semester

Vim基本介绍

Vim 是一个多模态编辑 器：它对于插入文字和操纵文字有不同的模式。

Vim 是可编程的（可以使用 Vimscript 或者像 Python 一样的其他程序语言），Vim 的接口本身也是一个程序语言：键入操作（以及其助记名） 是命令，这些命令也是可组合的。

多种操作模式：

- **正常模式**：在文件中四处移动光标进行修改
- **插入模式**：插入文本
- **替换模式**：替换文本
- **可视化模式**（一般，行，块）：选中文本块
- **命令模式**：用于执行命令

Vim 会在左下角显示当前的模式。Vim 启动时的默认模式是正常模式。通常你会把大部分 时间花在正常模式和插入模式。

<ESC>（退出键）从任何其他模式返回正常模式

在正常模式：

- 键入 i 进入插入 模式
- R进入替换模式
- v 进入可视（一般）模式
- V 进入可视（行）模式
- <C–v> （Ctrl–V, 有时也写作^V）进入可视（块）模式
- : 进入命令模式

Vim基本操作

插入文本

正常模式，键入 i 进入插入模式，直到你键入 <ESC> 返回正常模式。

缓存，标签页，窗口

Vim 会维护一系列打开的文件，称为“缓存”.

一个 Vim 会话包含一系列标签页，每个标签页包含 一系列窗口（分隔面板）。每个窗口显示一个缓存。

缓存和窗口不是一一对应的关系；窗口只是视角。一个缓存可以在多个窗口打开，甚至在同一个标签页内的多个窗口打开。

Vim 默认打开一个标签页，这个标签也包含一个窗口。

命令行

在正常模式下键入：进入命令行模式。 在键入：后，你的光标会立即跳到屏幕下方的命令行。

- q 退出（关闭窗口）
- w 保存（写）
- wq 保存然后退出
- e {文件名} 打开要编辑的文件
- ls 显示打开的缓存
- help {标题} 打开帮助文档
- ○ help : w 打开 :w 命令的帮助文档
- ○ help w 打开 w 移动的帮助文档

Vim的接口是一种编程语言

移动

- 基本移动: hjkl （左， 下， 上， 右）
- 词： w （下一个词）， b （词初）， e （词尾）
- 行： 0 （行初）， ^ （第一个非空格字符）， $ （行尾）
- 屏幕： H （屏幕首行）， M （屏幕中间）， L （屏幕底部）
- 翻页： Ctrl–u （上翻）， Ctrl–d （下翻）
- 文件： gg （文件头）， G （文件尾）
- 行数： :{行数}<CR> 或者 {行数}G ({行数}为行数)
- 杂项： % （找到配对，比如括号或者 /* */ 之类的注释对）
- 查找： f{字符}， t{字符}， F{字符}， T{字符}
- ○ 查找/到 向前/向后 在本行的{字符}
- ○ , / ; 用于导航匹配
- 搜索: /{正则表达式}, n / N 用于导航匹配

修饰语

你可以用修饰语改变“名词”的意义。修饰语有 i，表示“内部”或者“在内“，和 a， 表示”周围“。

- ci(改变当前括号内的内容
- ci[改变当前方括号内的内容
- da' 删除一个单引号字符串， 包括周围的单引号

Vim的接口是一种编程语言

编辑

- i 进入插入模式
- ○ 但是对于操纵/编辑文本，不单想用退格键完成
- O / o 在之上/之下插入行
- d{移动命令} 删除 {移动命令}
- ○ 例如，dw 删除词, d$ 删除到行尾, d0 删除到行头。
- c{移动命令} 改变 {移动命令}
- ○ 例如，cw 改变词
- ○ 比如 d{移动命令} 再 i
- x 删除字符（等同于 dl）
- s 替换字符（等同于 xi）
- 可视化模式 + 操作
- ○ 选中文字, d 删除 或者 c 改变
- u 撤销, <C–r> 重做
- y 复制 / “yank” （其他一些命令比如 d 也会复制）
- p 粘贴
- 更多值得学习的: 比如 ~ 改变字符的大小写

计数

- 3w 向前移动三个词
- 5j 向下移动5行
- 7dw 删除7个词

自定义Vim

Vim 由一个位于 ~/.vimrc 的文本配置文件（包含 Vim 脚本命令）

扩展Vim

可以使用内置的插件管理系统。只需要创建一个~/.vim/pack/vendor/start/ 的文件夹，然后把插件放到这里（比如通过 git clone）。

- ctrlp.vim: 模糊文件查找
- ack.vim: 代码搜索
- nerdtree: 文件浏览器
- vim-easymotion: 魔术操作

博客文章：搜索“best Vim plugins”

> **Shell**
>
> 如果你是一个 Bash 用户，用 `set -o vi`。如果你用 Zsh：`bindkey -v`。Fish 用 `fish_vi_key_bindings`。另外，不管利用什么 shell，你可以 `export EDITOR=vim`。 这是一个用来决定当一个程序需要启动编辑时启动哪个的环境变量。 例如， `git` 会使用这个编辑器来编辑 commit 信息。
>
> **Readline**
>
> 很多程序使用 GNU Readline 库来作为 它们的命令控制行界面。Readline 也支持基本的 Vim 模式， 可以通过在 `~/.inputrc` 添加如下行开启：
>
> ```
> set editing-mode vi
> ```
>
> 比如，在这个设置下，Python REPL 会支持 Vim 快捷键。
>
> **其他**
>
> 甚至有 Vim 的网页浏览快捷键 browsers, 受欢迎的有 用于 Google Chrome 的 Vimium 和用于 Firefox 的 Tridactyl。 你甚至可以在 Jupyter notebooks 中用 Vim 快捷键。 这个列表 中列举了支持类 vim 键位绑定的软件。

Vim 进阶

这里我们提供了一些展示这个编辑器能力的例子。我们无法把所有的这样的事情都教给你，但是你 可以在使用中学习。一个好的对策是: 当你在使用你的编辑器的时候感觉 “一定有更好的方法来做这个”， 那么很可能真的有：上网搜寻一下。

搜索和替换

`:s` （替换）命令（文档）。

- `%s/foo/bar/g`
  - 在整个文件中将 foo 全局替换成 bar
- `%s/\[.*\](\(.*\))/\1/g`
  - 将有命名的 Markdown 链接替换成简单 URLs

多窗口

- 用 `:sp` / `:vsp` 来分割窗口
- 同一个缓存可以在多个窗口中显示。

宏

- `q{字符}` 来开始在寄存器 `{字符}` 中录制宏
- `q` 停止录制
- `@{字符}` 重放宏
- 宏的执行遇错误会停止
- `{计数}@{字符}` 执行一个宏{计数}次
- 宏可以递归
  - 首先用 `q{字符}q` 清除宏
  - 录制该宏，用 `@{字符}` 来递归调用该宏 （在录制完成之前不会有任何操作）
- 例子：将 xml 转成 json (file)
  - 一个有 “name” / “email” 键对象的数组
  - 用一个 Python 程序?
  - 用 sed / 正则表达式
    - `g/people/d`
    - `%s/<person>/{/g`
    - `%s/<name>\(.*\)<\/name>/"name": "\1",/g`
    - ...
  - Vim 命令 / 宏
    - `Gdd`, `ggdd` 删除第一行和最后一行
    - 格式化最后一个元素的宏 （寄存器 `e`）
      - 跳转到有 `<name>` 的行
      - `qe^r"f>s": "<ESC>f<C"<ESC>q`
    - 格式化一个的宏
      - 跳转到有 `<person>` 的行
      - `qpS{<ESC>j@eA,<ESC>j@ejS},<ESC>q`
    - 格式化一个标签然后转到另外一个的宏
      - 跳转到有 `<person>` 的行
      - `qq@pjq`
    - 执行宏到文件尾
      - `999@q`
    - 手动移除最后的 `,` 然后加上 `[` 和 `]` 分隔符