# Vim学习笔记

在vim中输入`:h vimtutor`可以通往Vim向导

## 标记与符号含义说明（写作体例）

Vim操作可以类比于弹钢琴，既可以像演奏和弦的方式同时按下几个键触发，也可以类似演奏主旋律按顺序依次输入一个或者多个键组合形成一条命令（普通模式命令）。

下面是这两种方式的具体含义。

演奏主旋律

| 标记 | 含义 |
| --- | --- |
| `x` | 按一次`x` |
| `dw` | 依次按`d`、`w` |
| `dap` | 依次按`d`、`a`、`p` |

演奏和弦

| 标记 | 含义 |
| --- | --- |
| `<C-n>` | 同时按`<Ctrl>`和`n` |
| `g<C-]>` | 按`g`，然后同时按`<Ctrl>`和`]` |
| `<C-r>0` | 同时按`<Ctrl>`和`r`，然后按`0` |
| `<C-w><C-=>` | 同时按`<Ctrl>`和`w`，然后同时按`<Ctrl>`和`=` |

占位符

使用花括号表示一条命令后可以跟有效按键集合。下面是例子：

| 标记 | 含义 |
| --- | --- |
| `f{char}` | 按`f`，后面跟任意字符 |

| 标记 | 含义 |
| --- | --- |
| `{a-z} | 按`，后面跟任意小写字母 （左上角反引号） |
| m{a-zA-Z} | 按m，后面接任意大小写字母 |
| d{motion} | 按d，后面跟任意动作命令 |
| <C-r>{register} | 同时按<Ctrl>和r，后面跟一个寄存器地址 |

特殊按键

| 标记 | 含义 |
| --- | --- |
| <Esc> | 按退出键 |
| <CR> | 按回车键，也写作<Enter> |
| <Ctrl> | 按控制键 |
| <Tab> | 按制表键 |
| <Shift> | 按切换键 |
| <S-Tab> | 同时按<Shift>和<Tab>键 |
| <Up> | 按上光标键 |
| <Down> | 按下光标键 |
| <Space> | 按空格键 |

提示符

| 提示符 | 含义 |
| --- | --- |
| $ | 在外部shell执行命令行 |
| : | 用命令行模式执行一条Ex命令 在Ex命令中是隐含的（比如:q是退出隐含了我们要按下Enter键） |
| / | 用命令行模式执行正向查找 |

| 提示符 | 含义 |
| --- | --- |
| ? | 用命令行模式执行反向查找 |
| = | 用命令行模式对一个Vim脚本表达式求值 |

激活Vim内置插件的最小配置：

essential.vim

```
set nocompatible
filetype plugin on
```

### 书中源码文件下载地址

https://pragprog.com/titles/dnvim/source_code

# VViimm解决问题的方式

## 结识.命令

> *. 命令可以让我们重复上次的修改，它是Vim中的瑞士军刀*

理解 . 命令的强大必须意识到一次修改的单位可以是字符、整行，甚至整个文件。

x命令会删除光标下的字符，这时使用 . 命令会让Vim继续删除光标下的字符（相当于重复删除），我们可以使用u命令撤销上述修改，使得文档恢复初始状态。

dd命令也做删除操作，但它删除的是整行。如果后面接 . 命令，会继续重复“上次修改”删除当前行。

>G命令会增加当前行到文档末尾处的缩进层级。我们试着用 . 命令进行重复该操作。

效果如下：

```
# 一开始
Line one

Line two
Line three
Line four

#  在第二行使用>G，然后下移一行使用.，再下移再使用.  结果如下
Line one

    Line two
        Line three
            Line four
```

上述都是在普通模式下使用命令。就算进入插入模式，也会形成一次修改，从进入（按下`i`）到返回到普通模式（输入`<Esc>`），Vim会记录每一个按键操作。

**Vim**可以录制任意数目的按键操作，然后在以后重复执行它们。这让我们可以把最常重复的工作流程录制下来，并用一个按键重放它们。我们可以把`.`命令当成一个很小的宏（macro）。

## 一键双雕——复合命令

很多Vim的单键命令都可以被看成两个或多个其他命令的组合。下面是类似的一些例子，它们有什么共同点呢？

| 复合命令 | 等效的长命令 |
| --- | --- |
| C | c$ |
| S | ^c |
| I | ^i |
| A | $a |
| o | A |
| O | ko |
| s | cl |

当我们输入ko命令时，想想自己在干嘛，然后要意识到我们可以把它换成O命令。

这些命令的共同点是，它们全都会从普通模式切换到插入模式。这会对`.`命令有怎样的影响？？当我们足够熟练时，我们在执行一些常规的插入任务完全不用进行插入模式就可以用`.`命令快速重复之前操作。

s命令把两个操作合并为一个：它先删除光标下的字符，然后进入插入模式。f{char}命令让Vim查找下一处指定字符出现的位置，如果找到了，就把光标移到那里（参见:h f）。当我们输入f+时，光标会直接移到下一个+号所在的位置。使用;命令会重复查找上次f命令所查找的字符。这样可以实现快速的查找与替换。

## 执行、重复、回退

> *在面对重复性工作时，我们需要让移动动作和修改都能够重复，这样就达到了最佳的编辑模式。Vim会记住我们的操作，并使最常用的快捷键触手可及，以方便我们重复执行它们。*

除了.命令，有些命令能以其他的方式重复。@:可以用来重复任意Ex命令；我们也可以输入&命令来重复上次的:substitute命令（它本身就是一条Ex命令）。

这样操作起来确实很爽，但是一不小心摁错或者多摁有可能导致很糟糕的情况。所以当我们不小心做过头时，知道回退会很有帮助。

可重复的操作及如何回退

| 目的 | 操作 | 重复 | 回退 |
| --- | --- | --- | --- |
| 做出一个修改 | {edit} | . | u |
| 在行内查找下一个指定字符 | f{char}/t{char} | ; | , |
| 在行内查找上一个指定字符 | F{char}/T{char} | ; | , |
| 在文档中寻找下一处匹配项 | /pattern<CR> | n | N |
| 在文档中寻找上一处匹配项 | ?pattern<CR> | n | N |
| 执行替换 | :s/target/replacement | & | u |
| 执行一系列修改 | qx{changes}q | @x | u |

## 查找并手动替换

*命令可以对当前光标上的单词进行查找，使用n键可以跳到下一个匹配项。cw命令可以删除从光标位置到单词结尾间的字符，并进入插入模式。

当我们只需要替换部分单词时，可以利用上述命令，用*匹配，然后cw删除并进入插入模式，改好后退出，然后用n跳至其他需要修改的匹配项，使用.命令重复修改。

## 结识.（点）范式

用一键移动，另一键执行，没有比这更好的方式了。在以后的学习和使用中，我们会一次又一次看到这种编辑模式，我们成为.范式。

# 模式

Vim提供一个区分模式的用户界面，就是说在Vim中按键盘上的任意键所产生的结果可能会不一样，这取决于当前处于哪种模式。知道我们当前处于哪种模式并掌握切换它们，这是极为重要的。

## 普通模式

普通模式是Vim的默认状态，这跟其他编辑器差别很大，它们一般都处于类似Vim插入模式的状态中。

普通模式的强大，来源于它可以把操作符和动作命令结合在一起。

### 停顿时请移开画笔

画家在休息时不会把画笔放在画布上。对Vim而言也是这样，普通模式就是Vim的自然放松状态，其名字（Normal mode）已经寓示了这一点。

就像画家只花小部分时间涂色一样，程序员也只花一小部分时间编写代码。绝大多数时间用来思考、阅读，以及在代码中穿梭浏览。而且，修改也不一定要切换到插入模式，普通模式中，我们有众多工具可以利用。

### 把撤销单元切成块

> *在Vim中，我们自己可以控制撤销的粒度。*

`u`键会触发撤销命令，它会撤销最新的修改。一次修改可以是改变文档内文本的任意操作，其中包括在普通模式、可视模式以及命令行模式所触发的命令，而且一次修改也包括了在插入模式中输入或删除的文本（即`i`**{insert some text}**也是一次修改）。

在Vim中，我们可以控制撤销命令的粒度。只要我们控制好对`<Esc>`键的使用，就可以使撤销命令作用于单词、句子或段落。

那么，我们多久离开一次插入模式比较合适呢？当然，这是一个个人喜好的问题。作者本身提出了非常科学合理的建议：让每次可撤销对应一次思考过程，在每一句话的结尾停顿一下，想一想接下来写什么。这形成了自然的中断点，当准备好继续写作时，使用`A`命令，如果觉得写错了，使用`u`撤销。这样思路会被切分成条理清晰的块。（果然见底深刻啊）

> *当处于插入模式时，如果光标位于行尾的话，另起一行的最快方式是按`<CR>`。不过有时我更喜欢按`<Esc>o`，这是因为我有预感，也许在撤销时我想拥有更细的粒度。*
>
> *（作者原话，可以细细品鉴一番）*

Note: 在插入模式下移动光标（使用 ）会重置修改状态。

### 构建可重复的删除

如果光标在一个单词的末尾，我们该如何删除这个单词呢？

1. 反向删除：按`db`命令删除光标起始位置到单词开头内容，然后使用`x`键删除光标处字符。

2. 正向删除：b命令把光标跳至单词开头，使用dw命令删除整个单词。
3. 删除整个单词：使用更为精准的aw文本对象而不是动作命令。连用daw即可删除该单词，可以解读为`delete a word`便于记忆。

我们可以使用dbx,bdw以及daw三种方式来删除某一个单词。但如果碰到需要重复删除的情况，哪种方式更好呢？请思考一下。

- 反向删除方案执行.会重复删除一个字符，即.==x。这并没有什么价值。
- 正向删除方案中,b只是一次普通移动，因此如果后续执行.命令会重复dw，删除从光标位置到下一个单词开头的内容。不过如果我们执行这种方案时删除的是一行最后一个单词的话，那么就没有“下一个单词”这一说了。
- 最后一种方案只调用了daw操作。它不仅仅删除了单词，还会删除一个空格。因此.命令是会重复daw删除操作的。因此daw是最后的胜者。

可以看出，常常需要经常一番思考才能充分利用.命令。如果我们需要在几个地方做同样的小修改，就可以尝试构造修改，以便于.命令重复执行。

### 用次数做简单的算数运算

> *很多普通命令都可以带一个次数前缀，这样Vim就会尝试指定该命令执行的次数，而不是只执行一次（参见:h count）。*

<C-a>和<C-x>命令分别对数字执行加和减。不带次数时逐个加减，带次数前缀时可以用它们加减任意整数（这个前缀被加或被减）。比如，我们把光标移到字符5，执行10<C-a>就会变成15。

如果光标不在数字上，它在会当前行正向查找一个数字，如果找到就跳到那里执行上述操作。

### 能够重复，就别用次数

如果你要删除两个单词，有几种方式：d2w和2dw，以及dw.。你会选择哪一种？

如果想要更好地撤销，最好不要费脑子想次数。记住口诀：执行、重复、回退。

当然了，有必要时我们也使用次数，一般是不需要重复的操作时。

### 双剑合璧，天下无敌

操作符 + 动作命令 = 操作

`d{motion}`命令可以对一个字符（dl）、一个完整单词（daw）或一整个段落（dap）进行操作，它作用的范围由动作命令决定。`c{motion}`、`y{motion}`以及其他命令类似，统称为操作符。使用`:h operator`可以查看完整列表。

下表给出常见的操作符。

| 命令 | 用途 |
| --- | --- |
| c | 修改 |

| 命令 | 用途 |
| --- | --- |
| d | 删除 |
| y | 复制到寄存器 |
| g~ | 翻转大小写 |
| gu | 转换为小写 |
| gU | 转换为大写 |
| > | 增加缩进 |
| < | 减少缩进 |
| = | 自动缩进 |
| ! | 使用外部程序过滤{motion}所跨越的行 |

操作符与动作命令的结合形成了一种语法。学习新的动作命令及操作符，就像在学习Vim词汇一样。随着词汇量的增加，我们就能表达更多的想法。

比如`gUaw`将当前单词转换为大写，如果知道了`ap`动作命令，我们能通过`gUap`将整段文字变成大写。

**Vim**的语法只有一条额外规则，即当一个操作符命令被连续调用两次时，它会作用于当前行。所以**dd**删除当前行，而**>>**缩进当前行。**gU**命令是一种特殊情况，我们既可以用**gUgU**，也可以使用**gUU**。

扩展命令组合的威力

自定义操作符与已有动作协同工作

我们可以定义新的操作符，Tim Pope的commentary.vim插件提供了一个很好的例子，该插件为Vim所支持的编程语言增添了注释及取消注释的命令。

如果想创建自定义操作符，阅读文档`:h :map-operator`。

自定义动作命令与已有操作符协同工作

我们可以定义新的动作命令及文本对象来进一步增强缺省的动作命令集。

Kana Natsuno的textobj-entire插件是一个很好的例子，它为Vim增加了两种新的文本对象`ie`和`ae`，它们作用于整个文件。

如果想创建自定义动作命令，阅读文档`:h omap-info`。