# 第 01 章 VIM解决问题的方式：用一键移动，另一键执行

## . 命令是做什么的？

. 命令是一个微型的宏，它会重复“上次的修改”。

## 什么是修改？

修改分为两种：

• 普通模式下执行一次命令，如 x、dd 及 > 等算作一次修改；

• 而从进入插入模式起（a、i、A、o、O），直到返回普通模式为止（<Esc>），这一连串的操作也算作一次修改。

使用 . 命令，它会重新执行上一次的修改。

## 问题 1：减少重复 | 重复在行尾添加内容

为如下程序中每条语句结尾添加分号

var foo = 1

var bar = 'a'

var foobar = foo + bar

变成

var foo = 1;

var bar = 'a';

var foobar = foo + bar;

**解答：**

光标首先移动到第一行任意位置，

• A 移动光标至语句结尾，并进入插入模式

• ; 插入分号

• <Esc> 返回普通模式

• j. 移动光标到下一行，并重复执行上一次的修改(A;<Esc>)

## 问题 2：以退为进 | 重复给某个字符前后添加一个空格

给下列语句中的加号（+）前后各添加一个空格：

var foo = "method("+argument1+","+argument2+")";

变成

var foo = "method(" + argument1 + "," + argument2 + ")";

**解答：**

假设光标位于第一个加号之前

• f+ 查找并移动到当前行的+号处

• s 删除光标下的字符，然后进入插入模式

• 空格 插入空格

• + 插入加号

• 空格 插入空格

• <Esc> 返回普通模式

• ; 重复 f+ 命令

• . 重复上一次修改(空格s空格<Esc>）

## 问题 3：关于重复及回退

如何在行内查找下一个字符？如何重复它？如何回退它？

f{char} / t{char} ; ,

如何在行内查找上一个字符？如何重复它？如何回退它？

F{char} / T{char} ; ,

如何在文档中查找下一匹配项？如何重复它？如何回退它？

/pattern<CR> n N

如何在文档中查找上一匹配项？如何重复它？如何回退它？

?pattern<CR> n N

如何替换当前行文本？如何重复它？如何回退它？

:s/target/replacement & u

## 问题 4：如何重复查找指定文本并手动替换

当然可以执行全局替换：

:%s/pattern/replacement

但有些时候需要在替换前进行确认是否执行替换。例如以下文本：

We're waiting for content before the site can go live.

If you are content with this, let's go ahead with it.

We'll launch as soon as we have the content.

我们想把 content 用 copy 代替，但 content 还有 happy 的含义，所以不是所有 content 都能替换为 copy，替换前需要确认。

**解答：**

1. 使用 /pattern 或 \* 来查找目标单词 content，此时光标位于单词 content 的首字母 c 的位置。

2. cw（删除光标位置到单词结尾之间的字符，并进入插入模式）

3. 输入单词 copy

4. <Esc>

5. n（查找下一个 content）

6. .（重复上一次修改操作，即步骤 2 - 4，不含步骤 1）

## 问题 5：以上技巧的共同点是什么？

只需按一次键就能把光标移动到下一目标上。用一键移动，另一键执行。

# 第 02 章 普通模式

普通模式是 Vim 的自然放松状态。

普通模式的强大，很大程度上源于它可以把操作符与动作命令结合在一起。

## 技巧 7 停顿时请移开画笔

就像画家只花一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，而且当确实需要修改时，谁说一定要切换到插入模式才行？我们可以重新调整已有代码的格式，复制它们，移动其位置，或是删除它们。在普通模式中，我们有众多的工具可以利用。

## 技巧 8 把撤销单元切成块

撤销不一定是最后输入的词或字符，我们可以自己控制撤销的粒度。

u 键用于撤销最新的修改。一次修改可以是：

• 普通模式、可视模式、命令行模式中所出发的命令。

• i{insert some text}<Esc> 也是一次修改。

只要控制好对 <Esc> 的使用，就可以撤销单词、句子或段落。那么多久离开一次插入模式呢？我喜欢每一个“可撤销块”对应一次思考过程。比如，我经常每一句话停顿一下，不管停顿时间有多短，都是一个自然的中断点，该退出插入模式。当我准备好继续写时，就按 A 命令继续。

在插入模式中，如果光标位于行尾，那么另起一行的最快方式是按 <CR>，但如果按 <Esc>o，那么在撤销时会更方便。

注意：如果在插入模式中使用上下左右箭头键移动光标，将会产生一个新的撤销块。因为这相当于先 <Esc>，然后用 h、j、k、l 移动光标，只不过没有退出插入模式。

## 技巧 9 构造可重复的修改

当我们写下

The end is nigh

时，光标停留在字母 h 上。如果我们想删除 nigh，可以使用 daw（可以解读为 delete a word，作用是删除光标所在单词，不论光标位于该单词的哪个字母上），该命令不仅会删除单词 nigh，还会跟着删除一个空格，使光标最终停留在单词 is 的最后一个字符上。这样做的好处是，如果你想再删除单词 is，只需执行一次 . 命令。

## 技巧 10 用次数做简单的算术

将如下代码中的 0px 改为 -180px：

.blog { background-position: 10px 10px }

• 使光标位于该行代码数字 10 或其之前的任意位置。

• 190<C-x>

说明：

• <C-x> 使光标所在数值减 1

• <C-a> 使光标所在数值加 1

**问题：007 的后面是什么？**

Vim 把以 0 开头的数字解释为八进制值，所以 007 + 001 = 010。除非将

set nrformats=

加入 vimrc 里。

## 技巧 11 能用重复，就别用次数

cw 的作用是，删除文本，直到下一个单词。所以：

• 当光标位于空格上时，会删除该位置到下一个单词之前的所有空格，并进入插入模式。

• 当光标位于一个单词的某个字母上时，会删除那个字母直到该单词结尾的所有字母，不包括其后的任何空格，并进入插入模式。

d2w、2dw 与 dw. 三个命令是等效的。唯一的不同在于使用撤销命令 u 时效果不同。前两个会一次性恢复所删除的全部两个单词，最后一个每次只恢复一个被删除的单词。

## 技巧 12 双剑合璧

操作符 + 动作命令 = 操作

d{motion} 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，其作用范围取决于动作命令 {motion}。c{motion}、y{motion} 以及其他一些命令也类似，它们统称为操作符（operator）。（可以用 :h operator 来查阅完整列表，使用 :q 退出查阅）

• dw：删除光标位置直到下一个单词开头之间的文本及空格。

• daw：不论光标位于当前单词的哪个字符，都会删除当前单词即其后所有空格直到下一个单词。

**Vim 操作符：**

c 修改

d 删除

y 复制到寄存器

g~ 反转大小写

gu 转换为小写

gU 转换为大写

> 增加缩进

< 减小缩进

= 自动缩进

! 使用外部程序过滤{motion}所跨越的行

Vim 语法有一条额外规则，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 是删除当前行，>> 缩进当前行，gUgU 是当前行变大写，但也可以使用简化版的 gUU，同理，g~~ 改变当前行大小写。

整篇文档变大写：ggguG

• gg：光标定位至文档开头第一个字符

• gu：变小写

• G：直到文档末尾最后一个字符

# 第 03 章 插入模式

## 技巧 13 在插入模式中及时更正错误

插入模式下：

<C-h> 删除前一个字符

<C-w> 删除前一个单词

<C-u> 删除至行首

这些命令不是 Vim 插入模式独有的，在 bash shell 中也可以使用它们。

## 技巧 14 插入-普通模式

<C-o>

可以带你进入“插入-普通模式”，在这个模式中，你可以执行一个普通模式的命令（不可是 Ctrl 组合键，可以是 dd、daw、3k、j、yy、ZZ 纯字母组合的命令），然后 Vim 会立刻返回到插入模式。

<C-o>zz

当所编辑的行位于屏幕顶部或底部时，可以使用 <C-o>，然后结合 zz 命令，使当前行移动至屏幕中心，然后继续编辑。免去了退出返回普通模式，再进入插入模式的麻烦。

## 技巧 15 不离开插入模式，粘贴寄存器中的文本

下面是一段尚未完成的文本：

Practical Vim, by Drew Neil

Read Drew Neil’s

我们想把书名插到最后一行，以补全该行。假设光标此时位于第一行首字母 P 的位置，那么：

* yt,
* jA<space>
* <C-r>0
* .<Esc>

得到：

Practical Vim, by Drew Neil

Read Drew Neil’s Practical Vim.