



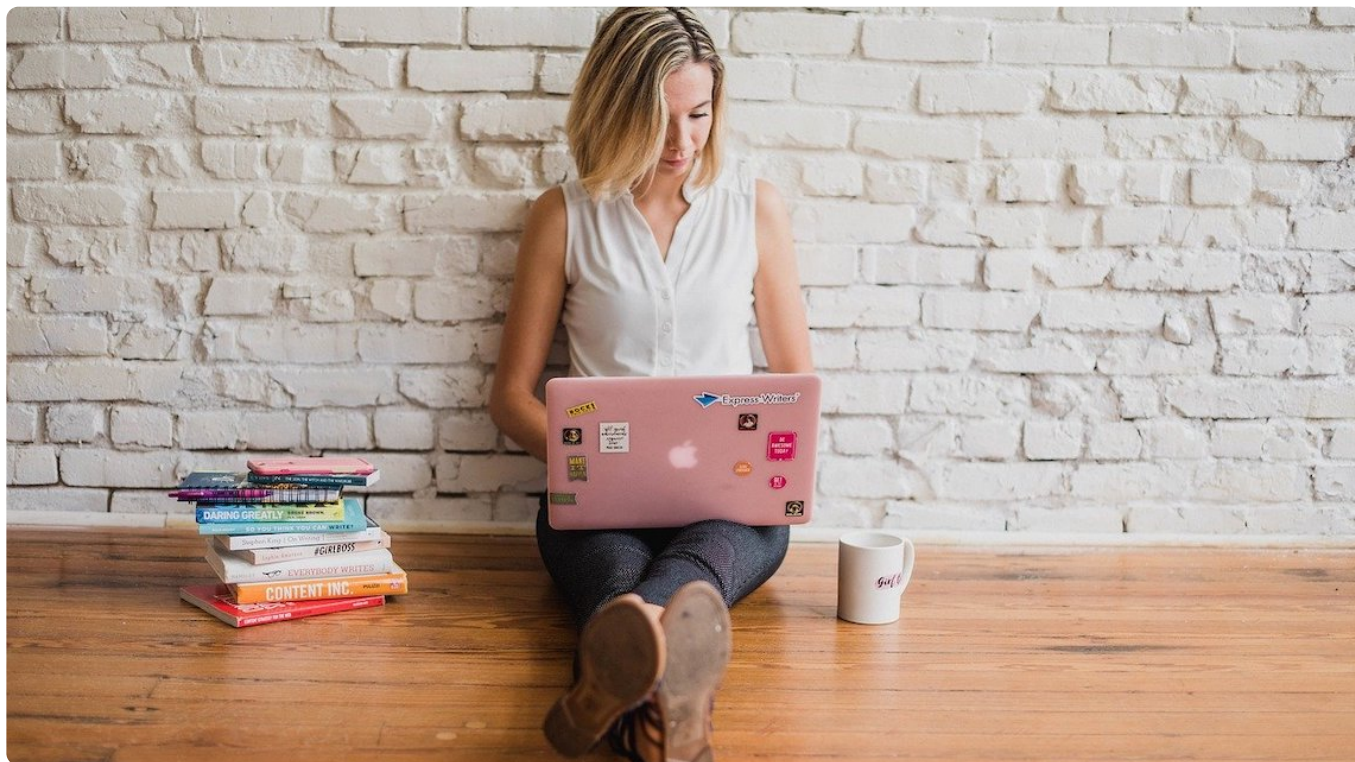
下载APP



03 | 更多常用命令：应对稍复杂的编辑任务

2020-07-29 吴咏炜

Vim 实用技巧必知必会

[进入课程 >](#)**讲述：吴咏炜**

时长 13:16 大小 12.16M



你好，我是吴咏炜。

上一讲我们通过 Vim 教程学习了 Vim 的基本命令，我还给你讲解了 Vim 的基本配置，现在你就已经可以上手基本的编辑工作了。

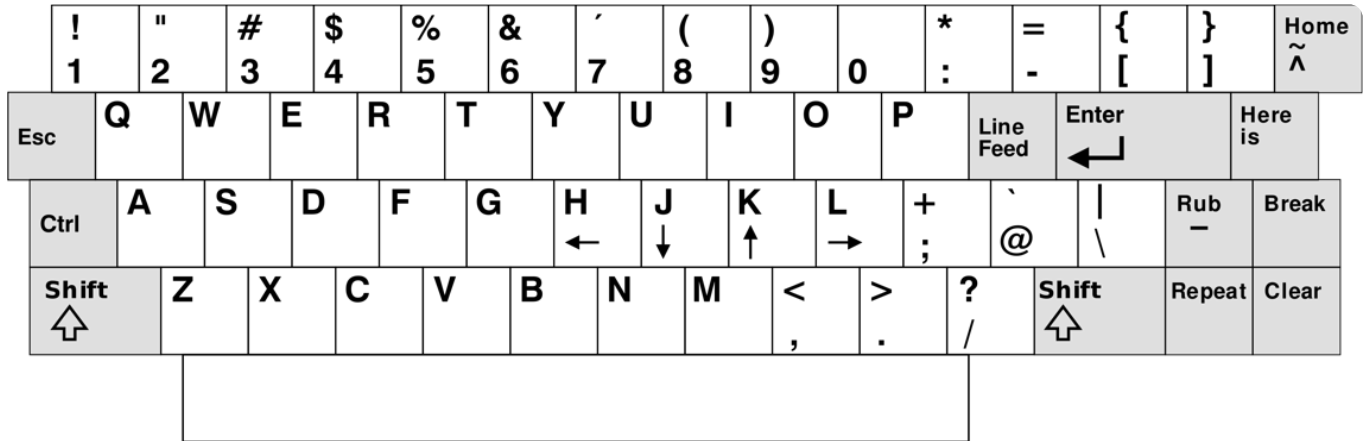
今天，我们将学习更多 Vim 的常用命令，以便更高效地进行编辑。我会先带你过一下**光标移动**命令和**文本修改**命令，然后重点讲解**文本对象**，随后快速讨论一下不能搭配文本修改的光标移动命令，最后讨论如何**重复**命令。



光标移动

我们先来讨论一下可以跟文本修改搭配的光标移动命令。

通过前面的课程，你已经知道，Vim 里的基本光标移动是通过 h、j、k、l 四个键实现的。之所以使用这四个键，是有历史原因的。你看一下 Bill Joy 开发 vi 时使用的键盘就明白了：这个键盘上没有独立的光标键，而四个光标符号直接标注在 H、J、K、L 四个字母按键上。



Lear Siegler ADM-3A 终端键盘的排布（图片源自维基百科）

当然，除了历史原因外，这四个键一直使用至今，还是有其合理性的。它们都处于打字机的本位排（home row）上，这样打字的时候，手指基本不用移动就可以敲击到。因此，即使到了键盘上全都有了光标移动键的今天，很多 Vim 的用户仍然会使用这四个键来移动光标。

不过，标准的光标移动键可以在任何模式下使用，而这四个键并不能在插入模式下使用，因此，它们并不构成完全的替代关系。

顺便提一句，你有没有注意到 ADM-3A 键盘上的 Esc 键在今天 Tab 的位置？在 Bill Joy 决定使用 Esc 来退出插入模式的时候，Esc 在键盘上的位置还没像今天那样跑到遥远的左上角去……

Vim 跳转到行首的命令是 0，跳转到行尾的命令是 \$，这两个命令似乎没什么特别的原因，一般用 <Home> 和 <End> 也没什么不方便的，虽然技术上它们有一点点小区别。如果你感兴趣、想进一步了解的话，可以参考帮助 [:help <Home>](#)。此外，我们也有 ^，用来跳转到行首的第一个非空白字符。

对于一次移动超过一个字符的情况，Vim 支持使用 b/w 和 B/W，来进行以单词为单位的跳转。它们的意思分别是 words Backward 和 Words forward，用来向后或向前跳转一个单词。小写和大写命令的区别在于，小写的跟编程语言里的标识符的规则相似，认为一个

单词是由字母、数字、下划线组成的（不严格的说法），而大写的命令则认为非空格字符都是单词。

```

                                *word*
A word consists of a sequence of letters, digits and underscores, or a
sequence of other non-blank characters, separated with white space (spaces,
tabs, <EOL>). This can be changed with the 'iskeyword' option. An empty line
is also considered to be a word.

                                *WORD*
A WORD consists of a sequence of non-blank characters, separated with white
space. An empty line is also considered to be a WORD.

~
~
~
~

```

小写 w 和大写 W 的区别

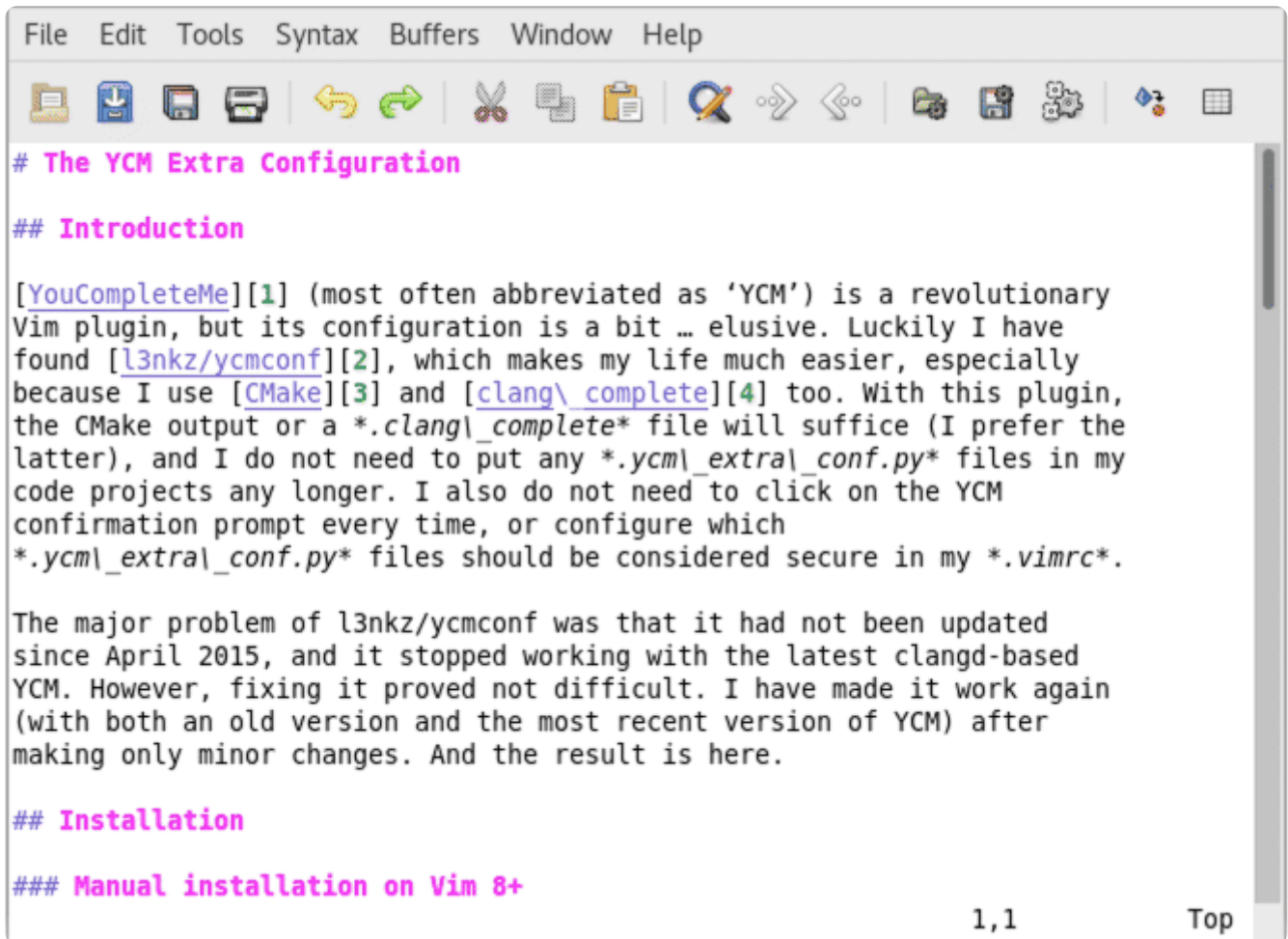
根据单个字符来进行选择也很常见。比如，现在光标在 `if (frame->fr_child != NULL)` 第五个字符上，如果我们想要修改括号里的所有内容，需要仔细考虑 `w` 的选词规则，然后输入 `c5w` 吗？这样显然不够方便。

这种情况下，我们就需要使用 `f` (`find`) 和 `t` (`till`) 了。它们的作用都是找到下一个（如果在输入它们之前先输入数字 n 的话，那就是下面第 n 个）紧接着输入的字符。两者的区别是，`f` 会包含这个字符，而 `t` 不会包含这个字符。在上面的情况下，我们用 `t` 就可以了：`ct`) 就可以达到目的。如果需要反方向搜索的话，使用大写的 `F` 和 `T` 就可以。

对于写文字的情况，比如给开源项目写英文的 README，下面的光标移动键也会比较有用：

(和) 移到上一句和下一句

{ 和 } 移到上一段和下一段



```
# The YCM Extra Configuration

## Introduction

[YouCompleteMe][1] (most often abbreviated as 'YCM') is a revolutionary
Vim plugin, but its configuration is a bit ... elusive. Luckily I have
found [l3nkz/ycmconf][2], which makes my life much easier, especially
because I use [CMake][3] and [clang\complete][4] too. With this plugin,
the CMake output or a *.clang\complete* file will suffice (I prefer the
latter), and I do not need to put any *.ycm\extra\conf.py* files in my
code projects any longer. I also do not need to click on the YCM
confirmation prompt every time, or configure which
*.ycm\extra\conf.py* files should be considered secure in my *.vimrc*.

The major problem of l3nkz/ycmconf was that it had not been updated
since April 2015, and it stopped working with the latest clangd-based
YCM. However, fixing it proved not difficult. I have made it work again
(with both an old version and the most recent version of YCM) after
making only minor changes. And the result is here.

## Installation

### Manual installation on Vim 8+
```

整句和整段的移动

在很多环境（特别是图形界面）里，Vim 支持使用 <C-Home> 和 <C-End> 跳转到文件的开头和结尾。如果遇到困难，则可以使用 vi 兼容的 gg 和 G 跳转到开头和结尾行（小区别：G 是跳转到最后一行的第一个字符，而不是最后一个字符）。

光标移动咱们就讲到这里。你需要重点掌握的就是 Vim 里除了简单的光标移动，还有“小词”、“大词”、句、段的移动，以及字符的搜索；每种方式都分向前和向后两种情况。

文本修改

接着，我们来看文本修改。

在 Vim 的教程里，我们已经学到，c 和 d 配合方向键，可以对文本进行更改。本质上，我们可以认为 c（修改）的功能就是执行 d（删除）然后 i（插入）。在 Vim 里，一般的原则就是，常用的功能，按键应尽可能少。因此很多相近的功能在 Vim 里会有不同的按键。不仅如此，大写键也一般会重载一个相近但稍稍不同的含义：

d 加动作来进行删除（dd 删除整行）；D 则相当于 d\$，删除到行尾。

c 加动作来进行修改（cc 修改整行）；C 则相当于 c\$，删除到行尾然后进入插入模式。

s 相当于 cl，删除一个字符然后进入插入模式；S 相当于 cc，替换整行的内容。

i 在当前字符前面进入插入模式；I 则相当于 ^i，把光标移到行首非空白字符上然后进入插入模式。

a 在当前字符后面进入插入模式；A 相当于 \$a，把光标移到行尾然后进入插入模式。

o 在当前行下方插入一个新行，然后在这行进入插入模式；O 在当前行上方插入一个新行，然后在这行进入插入模式。

r 替换光标下的字符；R 则进入替换模式，每次按键（直到 <Esc>）替换一个字符。

u 撤销最近的一个修改动作；U 撤销当前行上的所有修改。

熟练掌握这些按键需要一定的记忆和练习。但是，当你熟练掌握之后，大部分编辑操作只需要按一两个按键就能完成；而在你还没有做到熟练掌握之前，记住最简单、最有逻辑的按键也可以让你至少能够完成需要的编辑任务。

文本对象选择


好，接下来就是我们今天的重点内容，文本对象的选择了。我之所以把这部分内容作为这节课的重点，是因为这是一个很方便很强大的功能，并且特别适合程序中的逻辑块的编辑。

到现在，我们已经学习过，可以使用 c、d 加动作键对这个动作选定的文本块进行操作，也可以使用 v 加动作键来选定文本块（以便后续进行操作），我们也学习了好些移动光标的动作。不过，还有几个动作只能在 c、d、v 这样命令之后用，我们也需要学习一下。

这些选择动作的基本附加键是 a 和 i。其中，a 可以简单理解为英文单词 a，表示选定后续动作要求的完整内容，而 i 可理解为英文单词 inner，代表后续动作要求的内容的“内部”。这么说，还是有点抽象，我们来看一下具体的例子。

假设有下面的文本内容：

```
1 if (message == "sesame open")
```

 复制代码

我们进一步假设光标停在 “sesame” 的 “a” 上，那么：

`dw` (理解为 delete word) 会删除 `ame_` , 结果是 `if (message == "sesopen")`
`diw` (理解为 delete inside word) 会删除 `sesame` , 结果是 `if (message == "open")`
`daw` (理解为 delete a word) 会删除 `sesame_` , 结果是 `if (message == "open")`
`diW` 会删除 `"sesame` , 结果是 `if (message == open")`
`daW` 会删除 `"sesame_` , 结果是 `if (message == open")`
`di"` 会删除 `sesame open` , 结果是 `if (message == "")`
`da"` 会删除 `"sesame open"` , 结果是 `if (message ==)`
`di(或 di)` 会删除 `message == "sesame open"` , 结果是 `if (`
`da(或 da)` 会删除 `(message == "sesame open")` , 结果是 `if_`

上面演示了 `a`、`i` 和 `w`、双引号、圆括号搭配使用，这些对于任何语言的代码编辑都是非常有用的。实际上，可以搭配的还有更多：

搭配 `s` (sentence) 对句子进行操作——适合西文文本编辑

搭配 `p` (paragraph) 对段落进行操作——适合西文文本编辑，及带空行的代码编辑

搭配 `t` (tag) 对 HTML/XML 标签进行操作——适合 HTML、XML 等语言的代码编辑

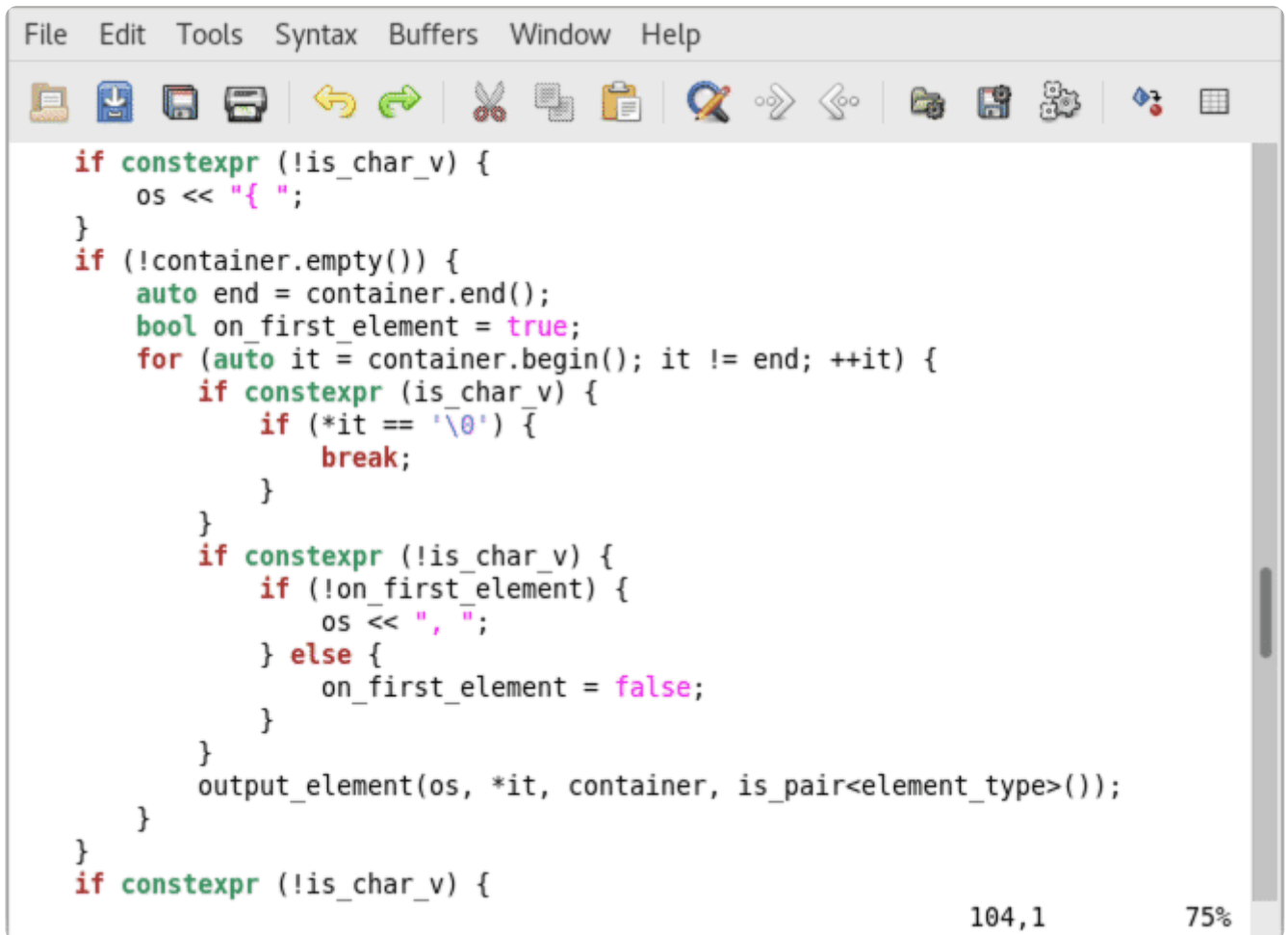
搭配 ``` 和 `'` 对这两种引号里的内容进行操作——适合使用这些引号的代码，如 `shell` 和 `Python`

搭配方括号 (`"["` 和 `"]"`) 对方括号里的内容进行操作——适合各种语言 (大部分都会用到方括号吧)

搭配花括号 (`"{"` 和 `"}"`) 对花括号里的内容进行操作——适合类 C 的语言

搭配角括号 (`"<"` 和 `">"`) 对角括号里的内容进行操作——适合 C++ 的模板代码

再进一步，在 `a` 和 `i` 前可以加上数字，对多个 (层) 文本对象进行操作。下面图中是一个示例：



```
File Edit Tools Syntax Buffers Window Help

if constexpr (!is_char_v) {
    os << "{ ";
}
if (!container.empty()) {
    auto end = container.end();
    bool on_first_element = true;
    for (auto it = container.begin(); it != end; ++it) {
        if constexpr (is_char_v) {
            if (*it == '\\0') {
                break;
            }
        }
        if constexpr (!is_char_v) {
            if (!on_first_element) {
                os << ", ";
            } else {
                on_first_element = false;
            }
        }
        output_element(os, *it, container, is_pair<element_type>());
    }
}
if constexpr (!is_char_v) {
```

104,1 75%

修改往上第 2 层花括号内的所有内容

你看，无论你使用什么语言，这些快捷的文本对象选择方式是不是总会有一种可以适用？我个人觉得这些功能绝对是 Vim 的强项了，所以，我再敲一次黑板，这部分内容是重点，不要嫌内容多，挨个儿用一用、练一练，你会发现这个功能非常实用，在写代码的时候常常会用得上。

更快地移动

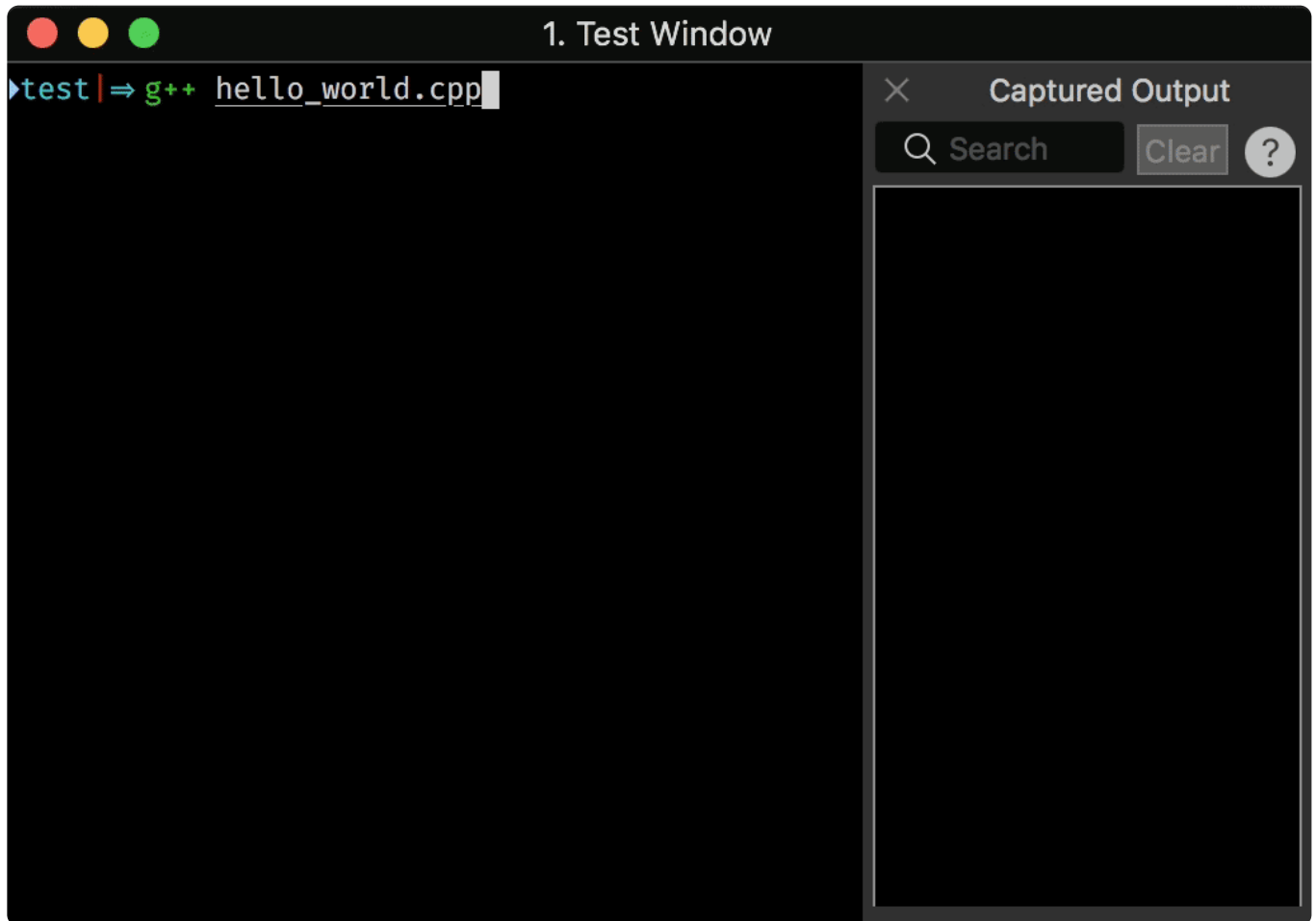
除了这讲开头提到的光标移动功能外，还有一些通常不和操作搭配的光标和屏幕移动功能。我们在这节里会快速描述一下。

我们仍然可以使用 <PageUp> 和 <PageDown> 来翻页，但 Vim 更传统的用法是 <C-B> 和 <C-F>，分别代表 Backward 和 Forward。

除了翻页，Vim 里还能翻半页，有时也许这种方式更方便，需要的键是 <C-U> 和 <C-D>，Up 和 Down。

如果你知道出错位置的行号，那你可以用数字加 G 来跳转到指定行。类似地，你可以用数字加 | 来跳转到指定列。这在调试代码的时候非常有用，尤其适合进行自动化。

下图中展示了 iTerm2 中  捕获输出并执行 Vim 命令的过程（用 `vim -c 'normal 5G36|'` 来执行跳转到出错位置第 5 行第 36 列）：



捕获错误信息并自动通过 Vim 命令行来跳转到指定位置

（如果你用 iTerm2 并对这个功能感兴趣，我设置的正则表达式是 `^([_a-zA-Z0-9+/.-]+):([0-9]+):([0-9]+):(?:fatal error|error|warning|note):`，捕获输出后执行的命令是 `echo "vim -c 'normal \2G\3|' \1"`。）

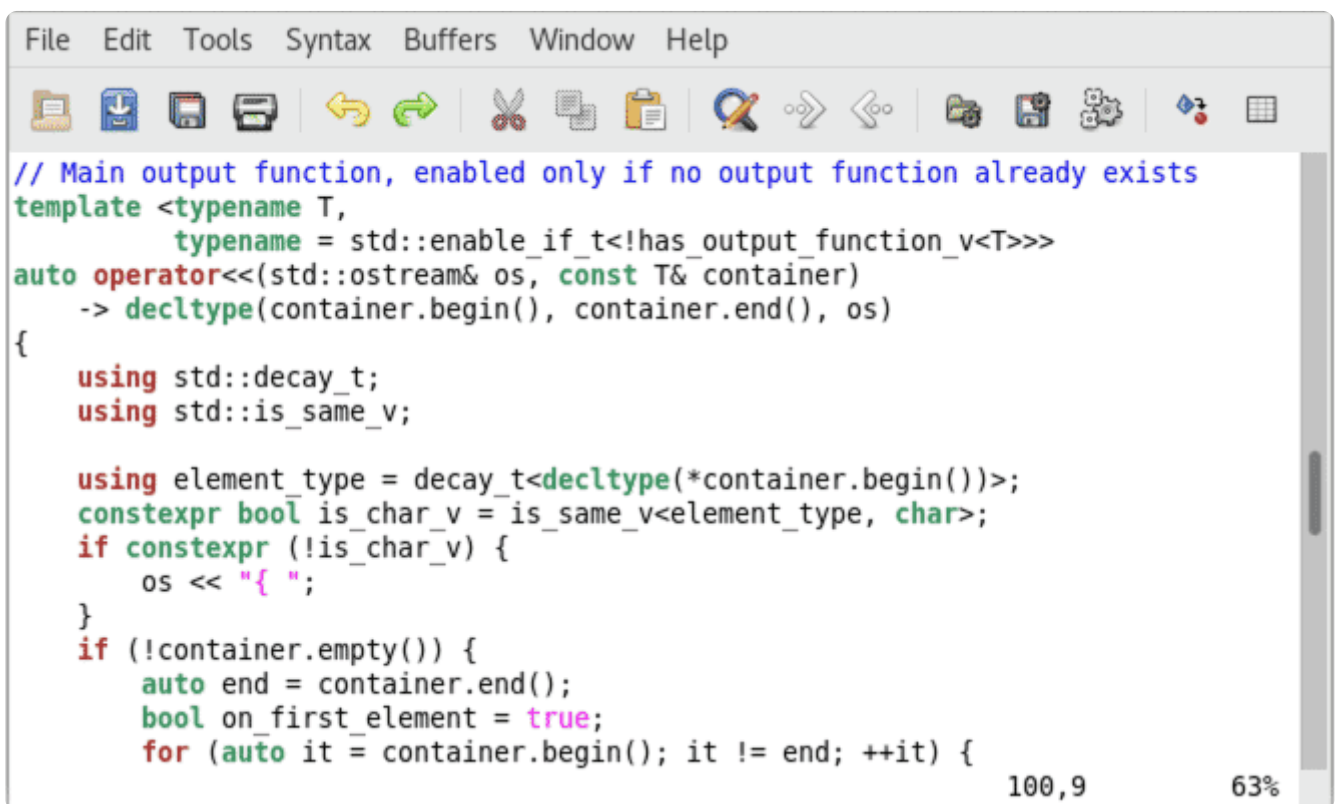
你只关心当前屏幕的话，可以快速移动光标到屏幕的顶部、中间和底部：用 H (High)、M (Middle) 和 L (Low) 就可以做到。

顺便提一句，`vimrc_example` 有一个设定，我不太喜欢：它会设 `set scrolloff=5`，导致只要屏幕能滚动，光标就移不到最上面的 4 行和最下面的 4 行里，因为一移进去屏幕就会自动滚动。这同样也会导致 H 和 L 的功能发生变化：本来是移动光标到屏幕的最上面和

最下面，现在则变成了移动到上数第 6 行和下数第 6 行，和没有这个设定时的 6H 与 6L 一样了。所以我一般会在 Vim 配置文件里设置 `set scrolloff=1`（你也可以考虑设成 0），减少这个设置的干扰。

只要光标还在屏幕上，你也可以滚动屏幕而不移动光标（不像某些其他编辑器，Vim 不允许光标在当前屏幕以外）。需要的按键是 `<C-E>` 和 `<C-Y>`。

另外一种可能更实用的滚动屏幕方式是，把当前行“滚动”到屏幕的顶部、中部或底部。Vim 里的对应按键是 `zt`、`zz` 和 `zb`。和上面的几个滚动相关的按键一样，它们同样受选项 `scrolloff` 的影响。



```
// Main output function, enabled only if no output function already exists
template <typename T,
        typename = std::enable_if_t!has_output_function_v<T>>>
auto operator<<(std::ostream& os, const T& container)
-> decltype(container.begin(), container.end(), os)
{
    using std::decay_t;
    using std::is_same_v;

    using element_type = decay_t<decltype(*container.begin())>;
    constexpr bool is_char_v = is_same_v<element_type, char>;
    if constexpr (!is_char_v) {
        os << "{ ";
    }
    if (!container.empty()) {
        auto end = container.end();
        bool on_first_element = true;
        for (auto it = container.begin(); it != end; ++it) {
```

光标移动和屏幕滚动

重复，重复，再重复

今天的最后，我来带你解决一个你肯定会遇到的问题，那就是如何更高效地解决重复的操作。

我们已经看到，在 Vim 里有非常多的命令，而且很多命令都需要敲好几个键。如果你要重复这样的命令，每次都要再手敲一遍，这显然是件很费力的事。作为追求高效率的编辑器，这当然是不可接受的。除了我们以后要学到的命令录制、键映射、自定义脚本等复杂操作外，Vim 对很多简单操作已经定义了重复键：

- ；重复最近的字符查找（f、t 等）操作
- ，重复最近的字符查找操作，反方向
- n 重复最近的字符串查找操作（/ 和 ?）
- N 重复最近的字符串查找操作（/ 和 ?），反方向
- 。重复执行最近的修改操作

有了这些，重复操作就非常简单了。要掌握它们的方法就是多练习，多用几次自然就会了。

内容小结

好了，今天的内容就讲完了，我们来做个小结。我们讨论了更多的一些常用 Vim 命令，包括：

基本光标移动命令（可配合 c、d 和 v）

文本修改命令小汇总

文本对象命令（c、d、v 后的 a 和 i）

更快的光标和屏幕移动功能

重复功能

今天讲的内容不难，重点是文本对象。你知道吗？我见到的 Vim 命令速查表里通常也没有它们，因而连很多 Vim 的老用户都不知道这些功能呢。所以，掌握了这部分内容，我们就已经走在很多 Vim 用户的前面了。请一定要多加练习，用好这个功能会大大提升你的代码编辑效率。

最后，提醒你去 GitHub 上看配置文件。配置文件我们有一处改动。类似地，适用于本讲的内容标签是 l3-unix 和 l3-windows。

课后练习

请把本讲里面描述的 Vim 功能自己练习一下，尤其需要重点掌握的是文本修改命令、文本对象命令和重复功能。其他某些功能可能只对部分人和某些场景有用，如果一个功能你觉

得用不上，不用去强记。毕竟，不用的功能，即使一时死记硬背可以记住，也很快会遗忘的。

欢迎你在留言区分享自己的学习收获和心得，有问题也要及时反馈，我们一起交流讨论。我们下一讲见！

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 基本概念和基础命令：应对简单的编辑任务

精选留言 (10)

写留言



我来也

2020-07-29

我的scrolloff好像配置到是1。

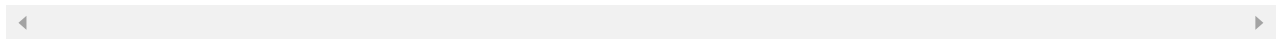
最早一直是0，不知道还可以调这玩意。

后来看网上别人设置的3，但体验后发现不太好，比较浪费空间。

但是这个还有那么一点作用，所以就调成了1。...

展开 ▾

作者回复: (之前的评论有问题，为避免误导，还是删除了。)



2

3



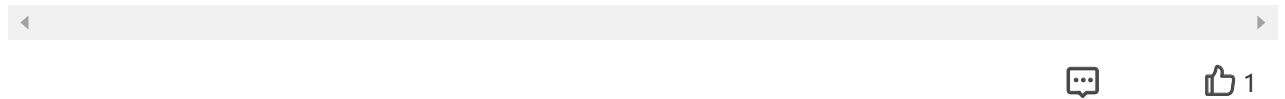
qinsi

2020-07-29

c2i{真是神操作，要是我的话就只会通过V模式先选中再修改了。选这门课也是希望能看到更多这样的操作，即在同样的情况下有经验的人是如何做的

展开 ▾

作者回复: 是的。这个是特色功能。



逗逼师父

2020-07-29

HHKB + VIM == 真香

展开 ▾



1



1+x

2020-07-29

太有收获了，支持吴老师😊

展开 ▾

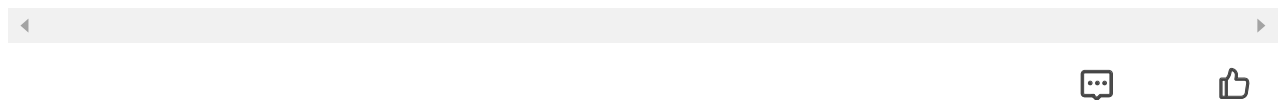


YouCompleteMe

2020-07-29

命令模式下的光标基于单词的移动，有什么好方法吗？感觉S-Left/Right比较麻烦。我一般都是^f切到命令历史，^c切回命令，对于输入一部分，发现有输入错误，想要修正比较麻烦

作者回复: 如果命令真复杂到这种程度，复制出来，编辑完，y\$，然后在命令行模式里 <C-r>" 如何？



25ma

2020-07-29

<https://github.com/25ma/vim-study-notes/blob/master/README.md> 按照老师文档练习了一遍，部分按照自己的语言理解的 如有不正确，还请大家帮我指正 谢谢



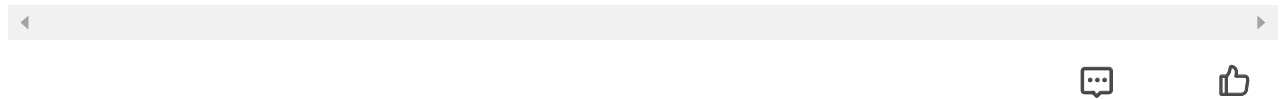
3.141516

2020-07-29

看到文本对象的操作真的是强大，但同时也有点复杂

展开 ▾

作者回复: 强大和复杂通常是挂钩的。🙄



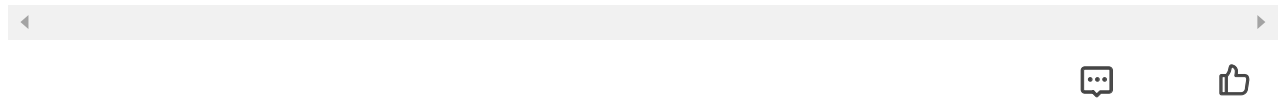
绝尘而去

2020-07-29

补充两个上下移动的键，gj和gk，这两个可以在由于屏幕限制而导致的换行中使用。

作者回复: 我没讲的功能永远有很多。🙄

这两个按键，我后面倒是会讲到（拓展1）。它们的主要功用是在很长的文本行里，而非一般的代码。



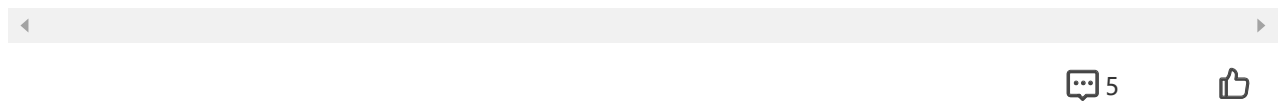
Captain perison

2020-07-29

老师您好，在插入模式下使用标准的光标移动键似乎不是很方便，或者退出到正常模式然后使用hjkl，频繁切换模式也不方便，这个有什么好的替代方法吗？谢谢！

展开 ▾

作者回复: 没更好的。移动少就光标移动键了。移动多、或后续操作不是插入，就回到正常模式。



我来也

2020-07-29

文本对象确实应该是vim中的神器，其他编辑器没有普通模式，估计是不好实现这个了。

文中的命令算是非常基础和全面的了。

w/b/e/ge/W/B/E/gE，0/^/\$和f/t/F/T在行内跳转，还是很灵活的。

...

展开 ▾

作者回复: vim-surround 我后面会介绍的。

