

The
Pragmatic
Programmers

以思维的速度来编辑文本

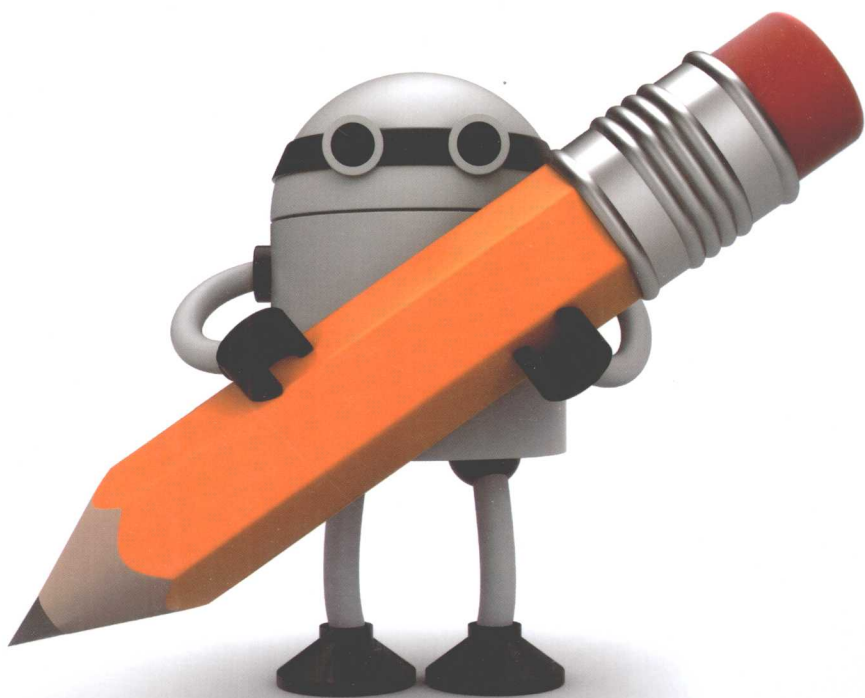
Vim 实用技巧

Practical Vim

Edit Text at the Speed of Thought

[英] Drew Neil 著

杨源 车文隆 译



 人民邮电出版社
POSTS & TELECOM PRESS

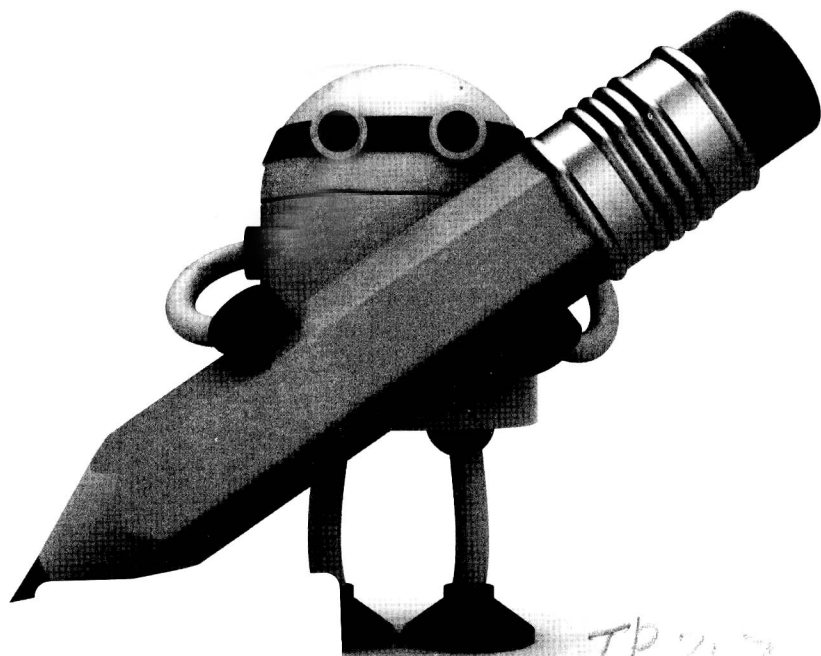
Vim实用技巧

Practical Vim

Edit Text at the Speed of Thought

[英] Drew Neil 著

杨源 车文隆 译



人民邮电出版社
北京

TP 317
-291

图书在版编目 (C I P) 数据

Vim实用技巧 / (英) 尼尔 (Neil,D.) 著 ; 杨源,
车文隆译. -- 北京 : 人民邮电出版社, 2014.5
ISBN 978-7-115-33869-3

I. ①V… II. ①尼… ②杨… ③车… III. ①
UNIX操作系统—程序设计 IV. ①TP316.81

中国版本图书馆CIP数据核字(2013)第284135号

版权声明

Simplified Chinese-language edition Copyright © 2014 by Posts & Telecom Press. All rights reserved.

Copyright © 2012 The Pragmatic Programmers, LLC. Original English language edition, entitled *Practical Vim*.

本书中文简体字版由 **The Pragmatic Programmers, LLC** 授权人民邮电出版社独家出版。未经出版者书面许可, 不得以任何方式复制或抄袭本书内容。

版权所有, 侵权必究。

-
- ◆ 著 [英] Drew Neil
译 杨 源 车文隆
责任编辑 陈冀康
责任印制 程彦红 杨林杰
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京艺辉印刷有限公司印刷
 - ◆ 开本: 800×1000 1/16
印张: 19.5
字数: 349千字 2014年5月第1版
印数: 1-3000册 2014年5月北京第1次印刷
- 著作权合同登记号 图字: 01-2012-8788号



定价: 59.00元

读者服务热线: (010)81055410 印装质量热线: (010)81055316
反盗版热线: (010)81055315

内容提要

Vim 是一款功能丰富而强大的文本编辑器，其代码补全、编译及错误跳转等方便编程的功能特别丰富，在程序员中得到非常广泛的使用。Vim 能够大大提高程序员的工作效率。对于 Vim 高手来说，Vim 能以与思考同步的速度编辑文本。同时，学习和熟练使用 Vim 又有一定的难度。

本书为那些想要提升自己的程序员编写，阅读本书是熟练地掌握高超的 Vim 技巧的必由之路。全书共 21 章，包括 121 个技巧。每一章都是关于某一相关主题的技巧集合。每一个技巧都有针对性地解决一个或一类问题，帮助读者提升 Vim 的使用技能。本书示例丰富，讲解清晰，采用一种简单的标记方法，表示交互式的编辑效果，可以帮助读者快速掌握和精通 Vim。

本书适合想要学习和掌握 Vim 工具的读者阅读，有一定 Vim 使用经验的程序员，也可以参考查阅以解决特定的问题。

读者对本书的评论

我通过本书学到的 Vim 知识比从其他渠道获得的要多得多。

➤ Robert Evans 软件工程师，编码狂人

读完本书的几章后，我意识到自己有多么孤陋寡闻，在 30 分钟时间里一下子被从中级用户打回到初学者。

➤ Henrik Nyh 软件工程师

本书不断地改变我对一个编辑器能做什么的信仰。

➤ John P. Daigle 开发人员，ThoughtWorks 公司

Drew 在本书中继续了他在 Vimcasts 网站上的杰出工作。对任何关注 Vim 的人来说，这都是一本不可错过的书。

➤ Anders Janmyr 开发人员，Jayway

本书在官方文档和如何真正使用 Vim 之间架设了一座跨越鸿沟的桥梁。读完几章以后，我就把默认编辑器换成了 Vim，从此再未换过。

➤ Javier Collado 自动化 QA 工程师，Canonical 公司

Drew Neil 远不止为我们展示了工作所用的正确工具。他于穿插叙述之中，揭示了每个决定背后的哲学。本书教你让 Vim 用手指思考，而不是期待你记住所有东西。

➤ Mislav Marohnic 顾问

我将 Vim 用于做服务器维护已经超过 15 年了，但只在最近才把它用于软件开发。我以为我了解 Vim，但本书极大地提升了我的编码效率。

➤ Graeme Mathieson 软件工程师，Rubaidh 公司

本书让我意识到对于 Vim 我还有多少东西要学。每个技巧都可以很容易地马上应用到工作过程当中，从多方面提升你的工作效率。

► Mathias Meyer 《Riak 手册》的作者

在 Vim 知识方面，本书是一个无尽的宝藏。现在我用 Vim 处理日常工作已经超过两年了，这本书给了我无穷的启发。

► Felix Geisendörfer 联合创始人，Transloadit

序

传统观点认为，Vim 的学习曲线很陡，但我相信绝大多数 Vim 用户对此不以为然。在学习 Vim 的初期，人们的确需要经历一段驼峰似的阻力，然而一旦完成了 vimtutor 的训练，并了解如何为 vimrc 配置一些基本选项后，就会达到一个新的高度，能用 Vim 完成实际工作了——尽管步履蹒跚，但终有回报。

接下来该做什么呢？来自互联网的答案是所谓的“技巧”——一种解决特定问题的灵丹妙药。当你觉得解决某个问题的方法不是最佳时，没准儿就要去搜索专门解决它的技巧了，或者你可能会主动看一些更受追捧的技巧。根据我的学习经验，这种策略的确奏效，不过这样学得很慢。“用 `g` 查找光标下的单词”这一招固然会让你受益匪浅，但却难以帮助你像 Vim 高手一样思考问题。

当我发现本书正是以这种“技巧”的方式组织章节时，你一定能理解我所持的怀疑态度。这区区上百条技巧怎么能让我举一反三呢？但当我翻了几页本书之后，我才意识到自己对“技巧”的理解太片面了。本书中所介绍的技巧与我认为是“问题 / 解决方法”方式有所不同，它旨在向人们传授如何像 Vim 高手一样思考问题。在某种意义上讲，这些技巧更像是寓言故事而非医师处方。书中的前几条技巧向人们介绍了应用范围很广的 `g` 命令，这是 Vim 高手们最重要的看家法宝，因为当时没人指点，我自己过了多年才意识到这一点。

正是由于这个原因，我才对本书的出版感到如此兴奋。如果现在再有 Vim 新手问我“下一步该学什么？”，我知道该告诉他们什么了。不管怎么说，本书甚至还教会了我不少东西呢。

Tim Pope

Vim 核心贡献者

2012 年 4 月

自序

本书是为那些想提升自己的程序员写的。你一定听说过，对于 Vim 高手来说，Vim 能以思考的速度编辑文本。阅读本书则是通往此途的必经之路。

本书是精通 Vim 的捷径。尽管它不会手把手教你，不过初学者可以先运行随 Vim 发布的交互式课程——Vim 向导^①来了解必备的知识。本书则在这一基础之上着重介绍核心概念，并为你讲解地道的用法。

Vim 是高度可配置的，然而定制是一件很个性化的事情，因此我试图避免建议什么应该放进你的 vimrc 里什么不应该。相反，本书关注的是 Vim 编辑器的核心功能。不管你是通过 SSH 登录远端服务器工作，还是在用本地安装了插件而增添了额外功能的 GVim，这些功能都永远在那儿。精通了 Vim 的核心功能，你就获得了一个可移植的、强大的文本编辑工具。

本书如何组织

这是一本按技巧进行组织的书，它被设计成不必从头读到尾（没错！在下一章开头，我会建议你直接跳到正文）。每一章都是关于某一相关主题的技巧集合，而每个技巧都讲解一个特定的实用功能。有些技巧自成一体，而有些技巧则依赖本书中其他地方的内容，这些有依赖关系的技巧会以交叉引用的形式呈现给大家，因此你可以轻松找到所有内容。

虽然整本书的进度安排不是先从入门开始，然后再到高级，但是每个独立章节中的内容都是按循序渐进的方式来组织的。缺乏经验的读者可能更愿意先浏览全书，只阅读每章的前几个技巧；而资深的读者可能会重点看每章中比较靠后的技巧，或是根据需要查阅本书。

^① http://vimdoc.sourceforge.net/html/doc/usr_01.html#vimtutor

关于示例的说明

在 Vim 中，对一件给定的任务，总能找到不止一种解决办法。例如，第 1 章里的所有问题都围绕 `▮` 命令进行设计，以便讲解 `▮` 命令的应用，不过这些问题也都可以用 `:substitute` 命令解决。

在阅读我的解决方法时，你自己也许会想：“难道用这种方法做不是更快吗？”可能你是对的！我的解决方法只是在讲解一种特定的技术，试图透过它们简单的外表，找出它与你日常所面临问题的相似之处，而这一点才是这些技巧可以帮你节省时间的地方。

先学会盲打，然后再学习 Vim

如果你要低头看着键盘打字，那学习 Vim 的好处不会立竿见影地显现出来。要高效地使用 Vim，必须学会盲打。

Vim 的祖先要追溯到经典的 UNIX 编辑器 vi 和 ed，参见技巧 27 中的“Vim（及其家族）的词源”部分，它们比鼠标及所有点击界面出现得都早，因此根本没有这类接口，所有操作都通过键盘完成。Vim 也是一样，在 Vim 中所有的操作也都可以通过键盘完成。对盲打人员来说，这意味着用 Vim 做任何事都能更快些。

致 谢

感谢 Bram Moolenaar 创造了 Vim，也感谢所有对 Vim 的开发做出贡献的人。这是一个永恒的软件，我期盼自己能与它一同成长。

感谢 Pragmatic Bookshelf 公司的每个人，正是你们的齐心协力才使这本书变得更好。特别感谢本书的项目编辑 Kay Keppler，感谢他教导我成为一名作者，并促使本书成形，而不在乎经历了多少的成长之痛以及我偶尔使性子。我也想感谢 David Kelly，感谢他对我与众不同的排版要求所做出的巧妙设计。

本书刚开始并没有打算按照技巧的方式组织章节，但 Susannah Pfalzer 发现采用这种格式会更好。重写这么多内容的确很痛苦，但做完之后，我头一次写出了让自己满意的初稿。Susannah 知道什么是最好的，感谢她分享了这一见解。

感谢 Dave Thomas 和 Andy Hunt 创办了 Pragmatic Bookshelf 公司。我没想过让他们出版商出版本书，并且我很荣幸本书能和其他书一起列在他们的书目中。

如果没有技术审阅人员，本书不可能出版。每一位技术审阅人都为之贡献了力量，并帮助本书成形。在此我想感谢 Adam McCrea、Alan Gardner、Alex Kahn、Ali Alwasity、Anders Janmyr、Andrew Donaldson、Angus Neil、Charlie Tanksley、Ches Martin、Daniel Bretoi、David Morris、Denis Gorin、Elyézer Mendes Rezende、Erik St. Martin、Federico Galassi、Felix Geisendörfer、Florian Vallen、Graeme Mathieson、Hans Hassel-berg、Henrik Nyh、Javier Collado、Jeff Holland、Josh Sullivan、Joshua Flanagan、Kana Natsuno、Kent Frazier、Luis Merino、Mathias Meyer、Matt Southerden、Mislav Marohnic、Mitch Guthrie、Morgan Prior、Paul Barry、Peter Aronoff、Peter Rihn、Philip Roberts、Robert Evans、Ryan Stenhouse、Steven! Ragnarök、Tibor Simic、Tim Chase、Tim Pope、Tim Tyrrell 以及 Tobias Sailer。

另外，我想感谢整个 Vim 社区，感谢他们通过互联网分享见解。通过阅读 StackOverflow 上 Vim 标签中的内容和订阅 vim_use 邮件列表，我学到了本书中的很多技巧。

Tim Pope 的 rails.vim 插件对于说服我皈依 Vim 起了很大的作用，并且他开发的许多其他插件也都成为我的 Vim 设置中必不可少的组成部分。我也从 Kana Natsuno 的插件中领悟到很多东西，在我印象里，他的自定义文本对象是对 Vim 核心功能最好的扩展。感谢你们两位把 Vim 这把“锯子”磨得更加锋利，使我们大家从中获益。

感谢 Joe Rozner 提供的 `wakeup` 源码，我使用它来介绍 `:make` 命令；感谢 Oleg Efimov 对 `nodeline` 缺陷的快速反馈，也感谢 Ben Cormack 对 `robots` 以及 `ninjas` 的解释。

2012 年 1 月，我们搬到了柏林，在这里的技术社区的启发下，我完成了本书。我想感谢 Gregor Schmidt 成立了 Vim 柏林用户组，也感谢 Schulz-Hofen 举办的招待我们的聚会。与 Vim 用户交谈的机会，真正帮我理清了思路，因此我感激每个参加 Vim 柏林会议的人。也感谢 Daniel 及 Nina Holle 把自己的房子转租给我们，它既是居住的绝佳场所，也是工作的高效环境。

2011 年 3 月，我需要动手术清除肠道粘连。不幸的是，我离家很远；幸运的是，我妻子陪伴在身边。Hannah 把我送到半岛南部医院，在那里我受到了很好的照顾。我想对那里的所有医务人员表示感谢，感谢他们对我的悉心照料，也感谢 Shawket Gerges 医生为我成功地进行了手术。

当我母亲知道我要动手术时，她抛下一切乘坐最早航班飞到埃及。要知道当时这个国家正在发生革命，她老人家需要多大的勇气才能做到这一切啊。我无法想象没有我母亲的支持与经验，我和 Hannah 该如何度过这段困难时期。我为一生中拥有两个如此伟大的女人而感到幸福。

写作体例说明

在本书中，我会通过图例进行演示，而不是描述它们。用书面语不太容易做到这一点。为了显示在交互式编辑会话中所采取的步骤，我采用了一种简单的标记方法，把按键操作及 Vim 缓冲区的内容排在一起进行说明。

如果你急于动手操作的话，现在可以直接跳过这一部分。这一部分主要描述本书沿用的体例，你会发现其中很多都不需要解释。或许在某个地方你会偶然发现一个符号，想搞清楚它究竟代表什么意思。当这种情况发生时，你可以回到这一部分来寻求答案。

了解 Vim 的内置文档

了解 Vim 文档的最好方式是花点时间进行阅读。我在书中给出了 Vim 文档入口的“超链接”，以方便读者找到相关文档。例如，这里有一个通往 Vim 向导的“超链接”：`:h vimguide`。

这个图标具有两个作用。第一，它起到指示牌的作用，把目光吸引到这些有用的参考信息上；第二，如果你在联网的电子设备上阅读本书的话，那么你可以点击这些图标，它会把你带到 Vim 在线文档的相应入口。从这个意义上讲，它的确是超链接。

但是，如果你正在阅读纸版书，那该怎么做？别担心，如果在手边有可访问的 Vim 程序，简单地输入图标前的命令即可。

例如，你可以输入：`:h vimguide`（`:h` 是 `:help` 命令的简写）。你可以把它想成 `vimguide` 文档的唯一地址，即某种形式的 URL。从这个意义上讲，此 `help` 引用也是一种指向 Vim 内置文档的超链接。

在书页中模拟 Vim 的标记

Vim 区分模式的界面把它同其他文本编辑器区别开来。以音乐作个比喻，让我们拿 Qwerty 键盘与钢琴键盘进行比较。一个钢琴家可以每次只弹一个琴键来演奏主旋律，他也可以一次弹多个键来演奏和弦。对于多数文本编辑器，要触发一个键盘快捷键，需要先按住一个或多个修饰键，如控制键或命令键，然后再按另外一个键，在 Qwerty

键盘上的这种操作方式，等同于在钢琴键盘上演奏和弦。

某些 Vim 命令也由演奏和弦的方式触发。不过普通模式命令则被设计成输入一串按键。在 Qwerty 键盘上的这种操作方式，则等同于在钢琴键盘上演奏主旋律。

`Ctrl-s` 是用来表示组合键命令的惯用约定，意为“同时按控制键及 `s` 键”，但这种约定方式并不适合用来描述 Vim 区分模式的命令集。在本节，我们将结识贯穿于全书的标记，在讲解 Vim 的用法时会用到它们。

演奏主旋律

在普通模式中，我们按次序输入一个或多个键组成一条命令。这些命令看起来像下面这样：

标记	含义
<code>x</code>	按一次 <code>x</code>
<code>dw</code>	依次按 <code>d</code> 、 <code>w</code>
<code>dap</code>	依次按 <code>d</code> 、 <code>a</code> 、 <code>p</code>

这些序列大多数包含两个或 3 个按键，但有的命令会更长。解读 Vim 普通模式命令序列的含义可能颇具挑战性，不过经过练习后你会做得更好。

演奏和弦

当你看到诸如 `<C-p>` 这样的键时，它的意思不是“先按 `<`，然后按 `C`，再按 `,`，等等”。`<C-p>` 标记等同于 `Ctrl-p`，意为“同时按 `<Ctrl>` 及 `p`”。

我不会无缘无故地选择这种标记方式的。首先，在 Vim 的文档中使用了这种标记（`:h key-notation` ①），我们也用它定义自定义按键映射项。另外，某些 Vim 命令由组合键及其他键以一定的次序组合在一起，这种标记也可以很好地表达这些命令。请看下面这些例子：

标记	含义
<code><C-n></code>	同时按 <code><Ctrl></code> 和 <code>n</code>
<code>g<C-l></code>	按 <code>g</code> ，然后同时按 <code><Ctrl></code> 和 <code>l</code>
<code><C-r>0</code>	同时按 <code><Ctrl></code> 和 <code>r</code> ，然后按 <code>0</code>
<code><C-w><C-=></code>	同时按 <code><Ctrl></code> 和 <code>w</code> ，然后同时按 <code><Ctrl></code> 和 <code>=</code>

占位符

很多 Vim 命令需要以一定的次序按两个或多个按键。有些命令后面必须跟某种特定类型的按键，而其他命令后面则可以跟键盘上的任意键。我使用花括号表示一条命令后可以跟有效按键集合。下面是一些例子：

标记	含义
<code>f{char}</code>	按 <code>f</code> ，后面跟任意字符
<code>[a-z]</code>	按 <code>[</code> ，后面跟任意小写字母
<code>m[a-zA-Z]</code>	按 <code>m</code> ，后面跟任意小写或大写字母
<code>d{motion}</code>	按 <code>d</code> ，后面跟任意动作命令
<code><C-r>{register}</code>	同时按 <code><Ctrl></code> 和 <code>r</code> ，后面跟一个寄存器地址

显示特殊按键

有些特殊按键以其名字表示，下表节选了其中的一些：

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

注意，空格由 `␣` 表示。它和 `f{char}` 命令组合在一起时记为 `f␣`。

区分不同模式下的输入

在操作 Vim 时，经常会从普通模式切换到插入模式，然后再切换回普通模式。Vim 中的每个键都可能具有不同的含义，这取决于当前哪个模式生效。我用了另一种样式表示在插入模式中输入的键，这可以让人很容易地把它们与普通模式下的按键区分开来。

看看这个例子 `cwreplacement<Esc>`。普通模式命令 `cw` 会删除从光标位置到当前词结尾处的文本，并切换到插入模式。然后我们在插入模式中输入单词“replacement”，并按 `<Esc>` 键再切换回普通模式。

普通模式所用的样式也用于可视模式，而插入模式的样式也用来表示命令行模式及替换模式下输入的按键。你可以通过上下文清楚地知道当前处于哪个模式。

在命令行中操作

在有些技巧中，我们会在 shell 或 Vim 中执行一条命令行命令。下面是在 shell 中执行 grep 命令的格式：

```
⇒$ grep -n Waldo *
```

下面是执行 Vim 内置的 `:grep` 命令的格式：

```
⇒:grep Waldo *
```

在全书中，`$` 符号表示在外部 shell 中执行一条命令行命令，而 `:` 提示符则表示这条命令在内部的命令行模式中执行。有时我们也会看到其他的提示符，包括：

提示符	含义
<code>\$</code>	在外部 shell 中执行命令行命令
<code>:</code>	用命令行模式执行一条 Ex 命令
<code>/</code>	用命令行模式执行正向查找
<code>?</code>	用命令行模式执行反向查找
<code>=</code>	用命令行模式对一个 Vim 脚本表达式求值

无论你何时在文中见到一条 Ex 命令，比如 `:write`，都可以假设我们按了 `<CR>` 键来执行该命令，否则该命令什么也不会做，因此你可以认为 `<CR>` 在 Ex 命令中是隐含的。

与之相反，Vim 的查找命令允许在按 `<CR>` 前预览第一个匹配项（参见技巧 81）。当你在文中见到一条查找命令时，比如 `/pattern<CR>`，你会看到 `<CR>` 键被显式地标出来了；如果 `<CR>` 被省略了，那是有意为之，也就是说你现在还不要按回车。

显示缓冲区内光标的位置

在显示缓冲区内容时，如果能指示当前光标位于何处，那会很有用。在下面的例子里，你可以看到光标位于单词“One”的第一个字母上：

One two three

当我们执行一项包含若干步的修改时，缓冲区的内容会经历一些中间状态。为了讲解这一过程，我使用了一个表格，在其左栏中显示所执行的命令，在右栏中显示缓冲区的内容。下面是个简单的例子：

按键操作	缓冲区内容
{start}	One two three
dw	two three

在第 2 行，我们运行 `dw` 命令删除了光标下的单词。通过查看位于同一行的缓冲区内容，我们可以立刻看到这条命令执行完后缓冲区的状态。

高亮显示查找匹配项

在讲解 Vim 的查找命令时，如果能把缓冲区内出现的每个匹配项都高亮显示出来，那会很有帮助。在下例中，查找字符串“the”会让出现该模式的 4 处地方被高亮显示出来：

按键操作	缓冲区内容
{start}	the problem with these new recruits is that they don't keep their boots clean.
/the<CR>	the problem with these new recruits is that they don't keep their boots clean.

你可以跳到技巧 80，了解如何激活 Vim 的查找高亮功能。

在可视模式中选择文本

可视模式允许我们在缓冲区内选择文本，然后在其上操作。在下例中，我们用 `it` 文本对象选中 `<a>` 标签内的文本：

按键操作	缓冲区内容
{start}	<code>Practical Vim</code>
vit	<code>Practical Vim</code>

注意，高亮显示可视选区的样式与高亮显示查找匹配项的样式是一样的。当你看到这种样式时，根据上下文就可以知道它究竟是代表一处查找匹配项，还是一个高亮选区。

下载本书中的示例

本书中的例子通常都先显示修改前的文件内容，并在示例文本中给出该文件所在的路径，如下所示：

```
macros/incremental.txt
partridge in a pear tree
turtle doves
French hens
calling birds
golden rings
```

每当你看到以这种方式列出的文件路径时，都表示该例可被下载。我建议你在 Vim 中打开此文件，然后亲自试试这个例子。这是学习 Vim 的最好方式。

要照着书中的例子操作，你可以从 Pragmatic Bookshelf 的网站上下载本书所有的示例和源代码^①。如果你在联网电子设备上阅读本书，也可以点击文件名来逐一获取每个文件。你可以用上面的例子试验一下。

使用 Vim 的出厂配置

Vim 是高度可配置的，如果你不喜欢其默认的行为，你可以改变它们。这本是好事，但是，如果你用自定义的 Vim 跟着做本书中的例子，可能会感到迷惑，你也许会发现有些东西并不像书中描述的那样工作。如果你怀疑是自定义配置造成了干扰，那么你可以做一个快速的测试。试着先退出 Vim，然后再用下列选项启动它：

```
⇒$ vim -u NONE -N
```

`-u NONE` 标志让 Vim 在启动时不加载你的 `vimrc`，这样，你的定制项就不会生效，插件也会被禁用。当用不加载 `vimrc` 文件的方式启动时，Vim 会切换到 `vi` 兼容模式，这将导致很多有用的功能被禁用，而 `-N` 标志则会启用 `'nocompatible'` 选项，防止进入 `vi` 兼容模式。

对于本书中的大多数例子来说，用 `vim -u NONE -N` 启动 Vim 应该可以确保你获得与书中的描述相符的体验，不过也有几处例外。有些 Vim 的内置功能是由 Vim 脚本实现的，也就是说，只有在激活插件时它们才会工作。下面的文件中包含了激活 Vim 内置插件的最小配置：

^① http://pragprog.com/titles/dnvim/source_code