第十三章 vi简介

There is an old joke about a visitor to New York City asking a passerby for directions to the city’s famous classical music venue:

Visitor: Excuse me, how do I get to Carnegie Hall?

Passerby: Practice, practice, practice!

有一个古老的笑话，说是一个在纽约的游客向行人打听这座城市中著名古典音乐场馆的方向：

游客： 请问一下，我怎样去卡内基音乐大厅？

行人： 练习，练习，练习!

Learning the Linux command line, like becoming an accomplished pianist, is not something that we pick up in an afternoon. It takes years of practice. In this chapter, we will introduce the vi (pronounced “vee eye”) text editor, one of the core programs in the Unix tradition. vi is somewhat notorious for its difficult user interface, but when we see a master sit down at the keyboard and begin to “play,” we will indeed be witness to some great art. We won’t become masters in this chapter, but when we are done, we will know how to play “chopsticks” in vi.

学习 Linux 命令行，就像要成为一名造诣很深的钢琴家一样，它不是我们一下午就能学会的技能。这需要 经历几年的勤苦练习。在这一章中，我们将介绍 vi（发音“vee eye”）文本编辑器，它是 Unix 传统中核心程序之一。 vi 因它难用的用户界面而有点声名狼藉，但是当我们看到一位大师坐在钢琴前开始演奏时，我们的确成了 伟大艺术的见证人。虽然我们在这里不能成为 vi 大师，但是当我们学完这一章后， 我们会知道怎样在 vi 中玩“筷子”。

Why We Should Learn vi

**为什么我们应该学习 vi**

In this modern age of graphical editors and easy-to-use text-based editors such as nano, why should we learn vi? There are three good reasons:

在现在这个图形化编辑器和易于使用的基于文本编辑器的时代，比如说 nano，为什么我们还应该学习 vi 呢？ 下面有三个充分的理由：

* vi is always available. This can be a lifesaver if we have a system with no graphical interface, such as a remote server or a local system with a broken X configuration. nano, while increasingly popular is still not universal. POSIX, a standard for program compatibility on Unix systems, requires that vi be present.
* vi 很多系统都预装。如果我们的系统没有图形界面，比方说一台远端服务器或者是一个 X 配置损坏了的本地系统，那么 vi 就成了我们的救星。虽然 nano 逐渐流行起来，但是它 还没有普及。POSIX，这套 Unix 系统中程序兼容的标准，就要求系统要预装 vi。
* vi is lightweight and fast. For many tasks, it’s easier to bring up vi than it is to find the graphical text editor in the menus and wait for its multiple megabytes to load. In addition, vi is designed for typing speed. As we shall see, a skilled vi user never has to lift his or her fingers from the keyboard while editing.
* vi 是轻量级且执行快速的编辑器。对于许多任务来说，启动 vi 比起在菜单中找到一个图形化文本编辑器， 再等待编辑器数倍兆字节的数据加载而言，要容易的多。另外，vi 是为了加快输入速度而设计的。 我们将会看到，当一名熟练的 vi 用户在编辑文件时，他或她的手从不需要移开键盘。
* We don’t want other Linux and Unix users to think we are sissies.
* 我们不希望其他 Linux 和 Unix 用户把我们看作胆小鬼。

Okay, maybe two good reasons.

好吧，可能只有两个充分的理由。

A Little Background

**一点儿背景介绍**

The first version of vi was written in 1976 by Bill Joy, a University of California at Berkley student who later went on to co-found Sun Microsystems. vi derives its name from the word “visual,” because it was intended to allow editing on a video terminal with a moving cursor. Previous to visual editors, there were line editors which operated on a single line of text at a time. To specify a change, we tell a line editor to go to a particular line and describe what change to make, such as adding or deleting text. With the advent of video terminals (rather than printer-based terminals like teletypes) visual editing became possible. vi actually incorporates a powerful line editor called ex, and we can use line editing commands while using vi.

第一版 vi 是在1976由 Bill Joy 写成的，当时他是加州大学伯克利分校的学生， 后来他共同创建了 Sun 微系统公司。vi 这个名字 来源于单词“visual”，因为它打算在带有可移动光标的视频终端上编辑文本。在发明可视化编辑器之前， 有一次只能操作一行文本的行编辑器。为了指定一个修改，我们告诉行编辑器到一个特殊行并且 说明做什么修改，比方说添加或删除文本。视频终端（而不是基于打印机的终端，像电传打印机）的出现 ，可视化编辑成为可能。vi 实际上整合了一个强大的叫做 ex 行编辑器, 所以我们在使用 vi 时能运行行编辑命令。

Most Linux distributions don’t include real vi; rather, they ship with an enhanced replacement called vim (which is short for “vi improved”) written by Bram Moolenaar. vim is a substantial improvement over traditional Unix vi and is usually symbolically linked (or aliased) to the name “vi” on Linux systems. In the discussions that follow, we will assume that we have a program called “vi” that is really vim.

大多数 Linux 发行版不包含真正的 vi；而是自带一款高级替代版本，叫做 vim（它是“vi improved”的简写）由 Bram Moolenaar 开发的。vim 相对于传统的 Unix vi 来说，取得了实质性进步。通常，vim 在 Linux 系统中是“vi”的符号链接（或别名）。 在随后的讨论中，我们将会假定我们有一个叫做“vi”的程序，但它其实是 vim。

Starting And Stopping vi

**启动和停止 vi**

To start vi, we simply type the following:

要想启动 vi，只要简单地输入以下命令：

[me@linuxbox ~]$ vi

And a screen like this should appear:

一个像这样的屏幕应该出现：

VIM - Vi Improved

....

Just as we did with nano earlier, the first thing to learn is how to exit. To exit, we enter the following command (note that the colon character is part of the command):

正如我们之前操作 nano 时，首先要学的是怎样退出 vi。要退出 vi，输入下面的命令（注意冒号是命令的一部分）：

:q

The shell prompt should return. If, for some reason, vi will not quit (usually because we made a change to a file that has not yet been saved), we can tell vi that we really mean it by adding an exclamation point to the command:

shell 提示符应该返回。如果由于某种原因，vi 不能退出（通常因为我们对文件做了修改，却没有保存文件）。 通过给命令加上叹号，我们可以告诉 vi 我们真要退出 vi。

:q!

Tip: If you get “lost” in vi, try pressing the Esc key twice to find your way again.

小贴示：如果你在 vi 中“迷失”了，试着按下 Esc 键两次来找到路（回到普通模式）。

Compatibility Mode

兼容模式

In the example startup screen above (taken from Ubuntu 8.04), we see the text “Running in Vi compatible mode.” This means that vim will run in a mode that is closer to the normal behavior of vi rather than the enhanced behavior of vim. For purposes of this chapter, we will want to run vim with its enhanced behavior. To do this, you have a few options:

上面实例中的启动屏幕（来自于 Ubuntu 8.04），我们看到一行文字“以 Vi 兼容的模式运行”。这意味着 vim 将以近似于 vi 常规的模式 运行，而不是 vim 的高级规范。为了这章的目的，我们想要使用 vim 的高级规范。要想这样做， 你有几个选择：

Try running vim instead of vi.

用 vim 来代替 vi。

If that works, consider adding alias vi=’vim’ to your .bashrc file.

如果命令生效，考虑在你的.bashrc 文件中添加别名 vi=’vim’。

Alternately, use this command to add a line to your vim configuration file:

或者，使用这个命令在你的 vim 配置文件中添加一行：

**echo “set nocp” >> ~/.vimrc**

Different Linux distributions package vim in different ways. Some distributions install a minimal version of vim by default that only supports a limiting set of vim features. While preforming the lessons that follow, you may encounter missing features. If this is the case, install the full version of vim.

不同的 Linux 发行版其 vim 软件包也迥然不同。一些发行版只是安装了 vim 的最小版本， 其默认只支持有限的 vim 特性。当练习随后的课程时，你可能会遇到缺失的功能。 如果是这种情况，就安装 vim 的完整版。

Editing Modes

**编辑模式**

Let’s start up vi again, this time passing to it the name of a nonexistent file. This is how we can create a new file with vi:

再次启动 vi，这次传递给 vi 一个不存在的文件名。这也是用 vi 创建新文件的方法。

[me@linuxbox ~]$ rm -f foo.txt

[me@linuxbox ~]$ vi foo.txt

If all goes well, we should get a screen like this:

如果一切运行正常，我们应该获得一个像这样的屏幕：

....

"foo.txt" [New File]

The leading tilde characters (”~”) indicate that no text exists on that line. This shows that we have an empty file. Do not type anything yet!

每行开头的波浪号（”~”）指示那一行不存在文本。这表示我们有一个空文件。还没有输入任何字符？

The second most important thing to learn about vi (after learning how to exit) is that vi is a modal editor. When vi starts up, it begins in command mode. In this mode, almost every key is a command, so if we were to start typing, vi would basically go crazy and make a big mess.

学习 vi 时，要知道的第二件非常重要的事情是（知道了如何退出 vi 后）vi 是一个模式编辑器。当 vi 启动后，进入 的是命令模式。这种模式下，几乎每个按键都是一个命令，所以如果我们打算输入字符，vi 会发疯，弄得一团糟。

Entering Insert Mode

**插入模式**

In order to add some text to our file, we must first enter insert mode. To do this, we press the “i” key. Afterwards, we should see the following at the bottom of the screen if vim is running in its usual enhanced mode (this will not appear in vi compatible mode):

为了在文件中添加文本，首先我们必须进入插入模式。按下”i”按键进入插入模式。之后，我们应该 在屏幕底部看到下面一行，如果 vi 运行在高级模式下（这不会出现在 vi 兼容模式下）：

-- INSERT --

Now we can enter some text. Try this:

现在我们能输入一些文本了。试着输入这些文本：

The quick brown fox jumped over the lazy dog.

To exit insert mode and return to command mode, press the Esc key.

按下 Esc 按键，退出插入模式并返回命令模式。

Saving Our Work

**保存我们的工作**

To save the change we just made to our file, we must enter an ex command while in command mode. This is easily done by pressing the “:” key. After doing this, a colon character should appear at the bottom of the screen:

为了保存我们刚才对文件所做的修改，我们必须在命令模式下输入一个 ex 命令。 通过按下”:”键，这很容易完成。按下冒号键之后，一个冒号字符应该出现在屏幕的底部：

:

To write our modified file, we follow the colon with a “w” then Enter:

为了写入我们修改的文件，我们在冒号之后输入”w”字符，然后按下回车键：

:w

The file will be written to the hard drive and we should get a confirmation message at the bottom of the screen, like this:

文件将会写入到硬盘，并且我们应该在屏幕底部得到一个确认信息，就像这样：

"foo.txt" [New] 1L, 46C written

Tip: If you read the vim documentation, you will notice that (confusingly) command mode is called normal mode and ex commands are called command mode. Beware.

小贴示：如果你阅读 vim 的文档，你注意到（混淆地）命令模式被叫做普通模式，ex 命令 叫做命令模式。当心。

Moving The Cursor Around

**移动光标**

While in command mode, vi offers a large number of movement commands, some of which it shares with less. Here is a subset:

当在 vi 命令模式下时，vi 提供了大量的移动命令，其中一些是与 less 阅读器共享的。这里 列举了一些：

|  |  |
| --- | --- |
| *Table 13-1: Cursor Movement Keys* | |
| Key | Move The Cursor |
| l or Right Arrow | Right one character. |
| h or Left Arrow | Left one character |
| j or Down Arrow | Down one line |
| k or Up Arrow | Up one line |
| 0 (zero) | To the beginning of the current line. |
| ^ | To the first non-whitespace character on the current line. |
| $ | To the end of the current line. |
| w | To the beginning of the next word or puntuation character. |
| W | To the beginning of the next word, ignoring puntuation character. |
| b | To the beginning of the previous word or punctuation character. |
| B | To the beginning of the previous word, ignoring punctuation characters. |
| Ctrl-f or Page Down | Down one page. |
| Ctrl-b or Page Up | Up one page. |
| numberG | To line number. For example, 1G moves to the first line of the file. |
| G | To the last line of the file. |

|  |  |
| --- | --- |
| *表13-1: 光标移动按键* | |
| 按键 | 移动光标 |
| l or 右箭头 | 向右移动一个字符 |
| h or 左箭头 | 向左移动一个字符 |
| j or 下箭头 | 向下移动一行 |
| k or 上箭头 | 向上移动一行 |
| 0 (零按键) | 移动到当前行的行首。 |
| ^ | 移动到当前行的第一个非空字符。 |
| $ | 移动到当前行的末尾。 |
| w | 移动到下一个单词或标点符号的开头。 |
| W | 移动到下一个单词的开头，忽略标点符号。 |
| b | 移动到上一个单词或标点符号的开头。 |
| B | 移动到上一个单词的开头，忽略标点符号。 |
| Ctrl-f or Page Down | 向下翻一页 |
| Ctrl-b or Page Up | 向上翻一页 |
| numberG | 移动到第 number 行。例如，1G 移动到文件的第一行。 |
| G | 移动到文件末尾。 |

Why are the h, j, k, and l keys used for cursor movement? Because when vi was originally written, not all video terminals had arrow keys, and skilled typists could use regular keyboard keys to move the cursor without ever having to lift their fingers from the keyboard.

为什么 h，j，k，和 l 按键被用来移动光标呢？因为在开发 vi 之初，并不是所有的视频终端都有 箭头按键，熟练的打字员可以使用规则的键盘按键来移动光标，他们的手从不需要移开键盘。

Many commands in vi can be prefixed with a number, as with the “G” command listed above. By prefixing a command with a number, we may specify the number of times a command is to be carried out. For example, the command “5j” causes vi to move the cursor down five lines.

vi 中的许多命令都可以在前面加上一个数字，比方说上面提到的”G”命令。在命令之前加上一个 数字，我们就可以指定命令执行的次数。例如，命令”5j”导致 vi 向下移动5行。

Basic Editing

**基本编辑**

Most editing consists of a few basic operations such as inserting text, deleting text and moving text around by cutting and pasting. vi, of course, supports all of these operations in its own unique way. vi also provides a limited form of undo. If we press the “u” key while in command mode, vi will undo the last change that you made. This will come in handy as we try out some of the basic editing commands.

大多数编辑工作由一些基本的操作组成，比如说插入文本，删除文本和通过剪切和粘贴来移动文本。 vi，当然，以它自己的独特方式来支持所有的操作。vi 也提供了有限的撤销形式。如果我们按下“u” 按键，当在命令模式下，vi 将会撤销你所做的最后一次修改。当我们试着执行一些基本的 编辑命令时，这会很方便。

Appending Text

**追加文本**

vi has several different ways of entering insert mode. We have already used the i command to insert text.

vi 有几种不同进入插入模式的方法。我们已经使用了 i 命令来插入文本。

Let’s go back to our foo.txt file for a moment:

让我们返回到我们的 foo.txt 文件中，呆一会儿：

The quick brown fox jumped over the lazy dog.

If we wanted to add some text to the end of this sentence, we would discover that the i command will not do it, since we can’t move the cursor beyond the end of the line. vi provides a command to append text, the sensibly named “a” command. If we move the cursor to the end of the line and type “a”, the cursor will move past the end of the line and vi will enter insert mode. This will allow us to add some more text:

如果我们想要在这个句子的末尾添加一些文本，我们会发现 i 命令不能完成任务，因为我们不能把 光标移到行尾。vi 提供了追加文本的命令，明智地命名为”a”命令。如果我们把光标移动到行尾，输入”a”, 光标就会越过行尾，vi 进入插入模式。这样就允许我们添加更多的文本：

The quick brown fox jumped over the lazy dog. It was cool.

Remember to press the Esc key to exit insert mode.

记住按下 Esc 按键来退出插入模式。

Since we will almost always want to append text to the end of a line, vi offers a shortcut to move to end of the current line and start appending. It’s the “A” command. Let’s try it and add some more lines to our file.

因为我们几乎总是想要在行尾附加文本，所以 vi 提供了一种快捷方式来移动到当前行的末尾，并且能添加 文本。它是”A”命令。试着用一下它，给文件添加更多行。

First, we’ll move the cursor to the beginning of the line using the “0” (zero) command. Now we type “A” and add the following lines of text:

首先，使用”0”(零)命令，将光标移动到行首。现在我们输入”A”，来添加以下文本行：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

Line 5

Again, press the Esc key to exit insert mode.

再一次，按下 Esc 按键退出插入模式。

As we can see, the “A” command is more useful as it moves the cursor to the end of the line before starting insert mode.

正如我们所看到的，大 A 命令非常有用，因为在启动插入模式之前，它把光标移到了行尾。

Opening A Line

**打开一行**

Another way we can insert text is by “opening” a line. This inserts a blank line between two existing lines and enters insert mode. This has two variants:

我们插入文本的另一种方式是“打开”一行。这会在存在的两行之间插入一个空白行，并且进入插入模式。 这种方式有两个变体：

|  |  |
| --- | --- |
| *Table 13-2: Line Opening Keys* | |
| Command | Opens |
| o | The line below the current line. |
| O | The line above the current line. |

|  |  |
| --- | --- |
| *表13-2: 文本行打开按键* | |
| 命令 | 打开行 |
| o | 当前行的下方打开一行。 |
| O | 当前行的上方打开一行。 |

We can demonstrate this as follows: place the cursor on “Line 3” then press the o key.

我们可以演示一下：把光标放到”Line 3”上，按下小 o 按键。

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

line 4

line 5

A new line was opened below the third line and we entered insert mode. Exit insert mode by pressing the Esc key. Press the u key to undo our change.

在第三行之下打开了新的一行，并且进入插入模式。按下 Esc，退出插入模式。按下 u 按键，撤销我们的修改。

Press the O key to open the line above the cursor:

按下大 O 按键在光标之上打开新的一行：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

Line 5

Exit insert mode by pressing the Esc key and undo our change by pressing u.

按下 Esc 按键，退出插入模式，并且按下 u 按键，撤销我们的更改。

Deleting Text

**删除文本**

As we might expect, vi offers a variety of ways to delete text, all of which contain one of two keystrokes. First, the x key will delete a character at the cursor location. x may be preceded by a number specifying how many characters are to be deleted. The d key is more general purpose. Like x, it may be preceded by a number specifying the number of times the deletion is to be performed. In addition, d is always followed by a movement command that controls the size of the deletion. Here are some examples:

正如我们期望的，vi 提供了各种各样的方式来删除文本，所有的方式包含一个或两个按键。首先， x 按键会删除光标位置的一个字符。可以在 x 命令之前带上一个数字，来指明要删除的字符个数。 d 按键更通用一些。类似 x 命令，d 命令之前可以带上一个数字，来指定要执行的删除次数。另外， d 命令之后总是带上一个移动命令，用来控制删除的范围。这里有些实例：

|  |  |
| --- | --- |
| *Table 13-3: Text Deletion Commands* | |
| Command | Deletes |
| x | The current character. |
| 3x | The current character and the next two character. |
| dd | The current line. |
| 5dd | The current line and the next four lines. |
| dW | From the cursor position to the beginning of the next word. |
| d$ | From the cursor position to the end of the current line. |
| d0 | From the cursor position to the beginning of the current line. |
| d^ | From the cursor position to the first non-whitespace character of the line. |
| dG | From the current line to the end of the file. |
| d20G | From the current line to the twentieth line of the file. |

|  |  |
| --- | --- |
| *表13-3: 文本删除命令* | |
| 命令 | 删除的文本 |
| x | 当前字符 |
| 3x | 当前字符及其后的两个字符。 |
| dd | 当前行。 |
| 5dd | 当前行及随后的四行文本。 |
| dW | 从光标位置开始到下一个单词的开头。 |
| d$ | 从光标位置开始到当前行的行尾。 |
| d0 | 从光标位置开始到当前行的行首。 |
| d^ | 从光标位置开始到文本行的第一个非空字符。 |
| dG | 从当前行到文件的末尾。 |
| d20G | 从当前行到文件的第20行。 |

Place the cursor on the word “It” on the first line of our text. Press the x key repeatedly until the rest of the sentence is deleted. Next, press the u key repeatedly until the deletion is undone.

把光标放到第一行单词“It”之上。重复按下 x 按键直到删除剩下的部分。下一步，重复按下 u 按键 直到恢复原貌。

Note: Real vi only supports a single level of undo. vim supports multiple levels.

注意：真正的 vi 只是支持单层面的 undo 命令。vim 则支持多个层面的。

Let’s try the deletion again, this time using the d command. Again, move the cursor to the word “It” and press dW to delete the word:

我们再次执行删除命令，这次使用 d 命令。还是移动光标到单词”It”之上，按下的 dW 来删除单词：

The quick brown fox jumped over the lazy dog. was cool.

Line 2

Line 3

Line 4

Line 5

Press d$ to delete from the cursor position to the end of the line:

按下 d$删除从光标位置到行尾的文本：

The quick brown fox jumped over the lazy dog.

Line 2

Line 3

Line 4

Line 5

Press dG to delete from the current line to the end of the file:

按下 dG 按键删除从当前行到文件末尾的所有行：

~

....

Press u three times to undo the deletion.

连续按下 u 按键三次，来恢复删除部分。

Cutting, Copying And Pasting Text

**剪切，复制和粘贴文本**

The d command not only deletes text, it also “cuts” text. Each time we use the d command the deletion is copied into a paste buffer (think clipboard) that we can later recall with the p command to paste the contents of the buffer after the cursor or the P command to paste the contents before the cursor.

这个 d 命令不仅删除文本，它还“剪切”文本。每次我们使用 d 命令，删除的部分被复制到一个 粘贴缓冲区中（看作剪切板）。过后我们执行小 p 命令把剪切板中的文本粘贴到光标位置之后， 或者是大 P 命令把文本粘贴到光标之前。

The y command is used to “yank” (copy) text in much the same way the d command is used to cut text. Here are some examples combining the y command with various movement commands:

y 命令用来“拉”（复制）文本，和 d 命令剪切文本的方式差不多。这里有些把 y 命令和各种移动命令 结合起来使用的实例：

|  |  |
| --- | --- |
| *Table13- 4: Yanking Commands* | |
| Command | Copies |
| yy | The current line. |
| 5yy | The current line and the next four lines. |
| yW | From the current cursor position to the beginning of the next word. |
| y$ | From the current cursor location to the end of the current line. |
| y0 | From the current cursor location to the beginning of the line. |
| y^ | From the current cursor location to the first non- whitespace character in the line. |
| yG | From the current line to the end of the file. |
| y20G | From the current line to the twentieth line of the file. |

|  |  |
| --- | --- |
| *表13-4: 复制命令* | |
| 命令 | 复制的内容 |
| yy | 当前行。 |
| 5yy | 当前行及随后的四行文本。 |
| yW | 从当前光标位置到下一个单词的开头。 |
| y$ | 从当前光标位置到当前行的末尾。 |
| y0 | 从当前光标位置到行首。 |
| y^ | 从当前光标位置到文本行的第一个非空字符。 |
| yG | 从当前行到文件末尾。 |
| y20G | 从当前行到文件的第20行。 |

Let’s try some copy and paste. Place the cursor on the first line of the text and type yy to copy the current line. Next, move the cursor to the last line (G) and type p to paste the line below the current line:

我们试着做些复制和粘贴工作。把光标放到文本第一行，输入 yy 来复制当前行。下一步，把光标移到 最后一行（G），输入小写的 p 把复制的一行粘贴到当前行的下面：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

Line 5

The quick brown fox jumped over the lazy dog. It was cool.

Just as before, the u command will undo our change. With the cursor still positioned on the last line of the file, type P to paste the text above the current line:

和以前一样，u 命令会撤销我们的修改。光标仍然位于文件的最后一行，输入大写的 P 命令把 所复制的文本粘贴到当前行之上：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

The quick brown fox jumped over the lazy dog. It was cool.

Line 5

Try out some of the other y commands in the table above and get to know the behavior of both the p and P commands. When you are done, return the file to its original state.

试着执行上表中一些其他的 y 命令，了解小写 p 和大写 P 命令的行为。当你完成练习之后，把文件 恢复原样。

Joining Lines

**连接行**

vi is rather strict about its idea of a line. Normally, it is not possible to move the cursor to the end of a line and delete the end-of-line character to join one line with the one below it. Because of this, vi provides a specific command, J (not to be confused with j, which is for cursor movement) to join lines together.

vi 对于行的概念相当严格。通常，不可能把光标移到行尾，再删除行尾结束符（回车符）来连接 当前行和它下面的一行。由于这个原因，vi 提供了一个特定的命令，大写的 J（不要与小写的 j 混淆了， j 是用来移动光标的）把行与行之间连接起来。

If we place the cursor on line 3 and type the J command, here’s what happens:

如果我们把光标放到 line 3上，输入大写的 J 命令，看看发生什么情况：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3 Line 4

Line 5

Search And Replace

**查找和替换**

vi has the ability to move the cursor to locations based on searches. It can do this on both a single line or over an entire file. It can also perform text replacements with or without confirmation from the user.

vi 有能力把光标移到搜索到的匹配项上。vi 可以在单一行或整个文件中运用这个功能。 它也可以在有或没有用户确认的情况下实现文本替换。

Searching Within A Line

**查找一行**

The f command searches a line and moves the cursor to the next instance of a specified character. For example, the command fa would move the cursor to the next occurrence of the character “a” within the current line. After performing a character search within a line, the search may be repeated by typing a semicolon.

f 命令查找一行，移动光标到下一个所指定的字符上。例如，命令 fa 会把光标定位到同一行中 下一个出现的”a”字符上。在一行中执行了字符的查找命令之后，通过输入分号来重复这个查找。

Searching The Entire File

**查找整个文件**

To move the cursor to the next occurrence of a word or phrase, the / command is used. This works the same way as we learned earlier in the less program. When you type the / command a “/” will appear at the bottom of the screen. Next, type the word or phrase to be searched for, followed by the Enter key. The cursor will move to the next location containing the search string. A search may be repeated using the previous search string with the n command. Here’s an example:

移动光标到下一个出现的单词或短语上，使用 / 命令。这个命令和我们之前在 less 程序中学到 的一样。当你输入/命令后，一个”/”字符会出现在屏幕底部。下一步，输入要查找的单词或短语后， 按下回车。光标就会移动到下一个包含所查找字符串的位置。通过 n 命令来重复先前的查找。 这里有个例子：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

Line 5

Place the cursor on the first line of the file. Type:

把光标移动到文件的第一行。输入：

/Line

followed by the Enter key. The cursor will move to line 2. Next, type n and the cursor will move to line 3. Repeating the n command will move the cursor down the file until it runs out of matches. While we have so far only used words and phrases for our search patterns, vi allows the use of regular expressions, a powerful method of expressing complex text patterns. We will cover regular expressions in some detail in a later chapter.

然后键入回车。光标会移动到第二行。下一步，输入 n，光标移到第三行。重复这个 n 命令，光标会 继续向下移动直到遍历了所有的匹配项。虽然目前，我们只是使用了单词和短语来作为我们的查找 模式，但是 vi 允许使用正则表达式，一种强大的用来表示复杂文本模式的方法。我们将会在随后 的章节里面详尽地介绍正则表达式。

Global Search And Replace

**全局查找和替代**

vi uses an ex command to perform search and replace operations (called “substitution” in vi) over a range of lines or the entire file. To change the word “Line” to “line” for the entire file, we would enter the following command:

vi 使用 ex 命令来执行查找和替代操作（vi 中叫做“替换”）。把整个文件中的单词“Line”更改为“line”， 我们输入以下命令：

:%s/Line/line/g

Let’s break this command down into separate items and see what each one does:

我们把这个命令分解为几个单独的部分，看一下每部分的含义：

|  |  |
| --- | --- |
| Item | Meaning |
| : | The colon character starts an ex command. |
| % | Specifies the range of lines for the operation. % is a shortcut meaning from the first line to the last line. Alternately, the range could have been specified 1,5 (since our file is five lines long), or 1,$ which means “from line 1 to the last line in the file.” If the range of lines is omitted, the operation is only performed on the current line. |
| s | Specifies the operation. In this case, substitution (search and replace). |
| /Line/line | The search pattern and the replacement text. |
| g | This means “global” in the sense that the search and replace is performed on every instance of the search string in the line. If omitted, only the first instance of the search string on each line is replaced. |

|  |  |
| --- | --- |
| 条目 | 含义 |
| : | 冒号字符运行一个 ex 命令。 |
| % | 指定要操作的行数。% 是一个快捷方式，表示从第一行到最后一行。另外，操作范围也 可以用 1,5 来代替（因为我们的文件只有5行文本），或者用 1,$ 来代替，意思是 “ 从第一行到文件的最后一行。” 如果省略了文本行的范围，那么操作只对当前行生效。 |
| s | 指定操作。在这种情况下是，替换（查找与替代）。 |
| /Line/line | 查找类型与替代文本。 |
| g | 这是“全局”的意思，意味着对文本行中所有匹配的字符串执行查找和替换操作。如果省略 g，则 只替换每个文本行中第一个匹配的字符串。 |

After executing our search and replace command our file looks like this:

执行完查找和替代命令之后，我们的文件看起来像这样：

The quick brown fox jumped over the lazy dog. It was cool.

line 2

line 3

line 4

line 5

We can also specify a substitution command with user confirmation. This is done by adding a “c” to the end of the command. For example:

我们也可以指定一个需要用户确认的替换命令。通过添加一个”c”字符到这个命令的末尾，来完成 这个替换命令。例如：

:%s/line/Line/gc

This command will change our file back to its previous form; however, before each substitution, vi stops and asks us to confirm the substitution with this message:

这个命令会把我们的文件恢复先前的模样；然而，在执行每个替换命令之前，vi 会停下来， 通过下面的信息，来要求我们确认这个替换：

replace with Line (y/n/a/q/l/^E/^Y)?

Each of the characters within the parentheses is a possible choice as follows:

括号中的每个字符都是一个可能的选择，如下所示：

|  |  |
| --- | --- |
| *Table 13-5: Replace Confirmation Keys* | |
| Key | Action |
| y | Perform the substitution. |
| n | Skip this instance of the pattern. |
| a | Perform the substitution on this and all subsequent instances of the pattern. |
| q or esc | Quit the substitution. |
| l | Perform this substitution and then quit. Short for"last". |
| Ctrl-e, Ctrl-y | Scroll down and scroll up, respectively. Useful for viewing the context of the proposed substitution. |

|  |  |
| --- | --- |
| *表13-5: 替换确认按键* | |
| 按键 | 行为 |
| y | 执行替换操作 |
| n | 跳过这个匹配的实例 |
| a | 对这个及随后所有匹配的字符串执行替换操作。 |
| q or esc | 退出替换操作。 |
| l | 执行这次替换并退出。l 是 “last” 的简写。 |
| Ctrl-e, Ctrl-y | 分别是向下滚动和向上滚动。用于查看建议替换的上下文。 |

If you type y, the substitution will be performed, n will cause vi to skip this instance and move on to the next one.

如果你输入 y，则执行这个替换，输入 n 则会导致 vi 跳过这个实例，而移到下一个匹配项上。

Editing Multiple Files

**编辑多个文件**

It’s often useful to edit more than one file at a time. You might need to make changes to multiple files or you may need to copy content from one file into another. With vi we can open multiple files for editing by specifying them on the command line:

同时能够编辑多个文件是很有用的。你可能需要更改多个文件或者从一个文件复制内容到 另一个文件。通过 vi，我们可以打开多个文件来编辑，只要在命令行中指定要编辑的文件名。

vi file1 file2 file3...

Let’s exit our existing vi session and create a new file for editing. Type :wq to exit vi saving our modified text. Next, we’ll create an additional file in our home directory that we can play with. We’ll create the file by capturing some output from the ls command:

我们先退出已经存在的 vi 会话，然后创建一个新文件来编辑。输入:wq 来退出 vi 并且保存了所做的修改。 下一步，我们将在家目录下创建一个额外的用来玩耍的文件。通过获取从 ls 命令的输出，来创建这个文件。

[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt

Let’s edit our old file and our new one with vi:

用 vi 来编辑我们的原文件和新创建的文件：

[me@linuxbox ~]$ vi foo.txt ls-output.txt

vi will start up and we will see the first file on the screen:

vi 启动，我们会看到第一个文件显示出来：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

Line 5

Switching Between Files

**文件之间转换**

To switch from one file to the next, use this ex command:

从这个文件转到下一个文件，使用这个 ex 命令：

:n

To move back to the previous file use:

回到先前的文件使用：

:N

While we can move from one file to another, vi enforces a policy that prevents us from switching files if the current file has unsaved changes. To force vi to switch files and abandon your changes, add an exclamation point (!) to the command.

当我们从一个文件移到另一个文件时，如果当前文件没有保存修改，vi 会阻止我们转换文件， 这是 vi 强制执行的政策。在命令之后添加感叹号，可以强迫 vi 放弃修改而转换文件。

In addition to the switching method described above, vim (and some versions of vi) also provide some ex commands that make multiple files easier to manage. We can view a list of files being edited with the :buffers command. Doing so will display a list of the files at the bottom of the display:

另外，上面所描述的转换方法，vim（和一些版本的 vi）也提供了一些 ex 命令，这些命令使 多个文件更容易管理。我们可以查看正在编辑的文件列表，使用:buffers 命令。运行这个 命令后，屏幕顶部就会显示出一个文件列表：

:buffers

1 # "foo.txt" line 1

2 %a "ls-output.txt" line 0

Press ENTER or type command to continue

To switch to another buffer (file), type :buffer followed by the number of the buffer you wish to edit. For example, to switch from buffer 1 which contains the file foo.txt to buffer 2 containing the file ls-output.txt we would type this:

要切换到另一个缓冲区（文件），输入 :buffer, 紧跟着你想要编辑的缓冲器编号。比如，要从包含文件 foo.txt 的1号缓冲区切换到包含文件 ls-output.txt 的2号缓冲区，我们会这样输入：

:buffer 2

and our screen now displays the second file.

我们的屏幕现在会显示第二个文件。

Opening Additional Files For Editing

**打开另一个文件并编辑**

It’s also possible to add files to our current editing session. The ex command :e (short for “edit”) followed by a filename will open an additional file. Let’s end our current editing session and return to the command line.

在我们的当前的编辑会话里也能添加别的文件。ex 命令 :e (编辑(edit) 的简写) 紧跟要打开的文件名将会打开 另外一个文件。 让我们结束当前的会话回到命令行。

Start vi again with just one file:

重新启动vi并只打开一个文件

[me@linuxbox ~]$ vi foo.txt

To add our second file, enter:

要加入我们的第二个文件，输入：

:e ls-output.txt

And it should appear on the screen. The first file is still present as we can verify:

它应该显示在屏幕上。 我们可以这样来确认第一个文件仍然存在：

:buffers

1 # "foo.txt" line 1

2 %a "ls-output.txt" line 0

Press ENTER or type command to continue

Note: You cannot switch to files loaded with the :e command using either the :n or :N command. To switch files, use the :buffer command followed by the buffer number.

注意：你不同通过:n 或:N 命令在由:e 命令加载的文件之间进行切换。这时要使用:buffer 命令， 其后加上缓冲区号码，来转换文件。

Copying Content From One File Into Another

**从一个文件复制内容到另一个文件**

Often while editing multiple files, we will want to copy a portion of one file into another file that we are editing. This is easily done using the usual yank and paste commands we used earlier. We can demonstrate as follows. First, using our two files, switch to buffer 1 (foo.txt) by entering:

当我们编辑多个文件时，经常地要复制文件的一部分到另一个正在编辑的文件。使用之前我们学到的 拉（yank）和粘贴命令，这很容易完成。说明如下。以打开的两个文件为例，首先转换到缓冲区1（foo.txt） ，输入：

:buffer 1

which should give us this:

我们应该得到以下输出：

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

Line 5

Next, move the cursor to the first line, and type yy to yank (copy) the line.

下一步，把光标移到第一行，并且输入 yy 来复制这一行。

Switch to the second buffer by entering:

转换到第二个缓冲区，输入：

:buffer 2

The screen will now contain some file listings like this (only a portion is shown here):

现在屏幕会包含一些文件列表（这里只列出了一部分）：

total 343700

-rwxr-xr-x 1 root root 31316 2007-12-05 08:58 [

....

Move the cursor to the first line and paste the line we copied from the preceding file by typing the p command:

移动光标到第一行，输入 p 命令把我们从前面文件中复制的一行粘贴到这个文件中：

total 343700

The quick brown fox jumped over the lazy dog. It was cool.

-rwxr-xr-x 1 root root 31316 2007-12-05 08:58 [

....

Inserting An Entire File Into Another

**插入整个文件到另一个文件**

It’s also possible to insert an entire file into one that we are editing. To see this in action, let’s end our vi session and start a new one with just a single file:

也有可能把整个文件插入到我们所编辑的文件中。看一下实际操作，结束 vi 会话，重新 启动一个只打开一个文件的 vi 会话：

[me@linuxbox ~]$ vi ls-output.txt

We will see our file listing again:

再一次看到我们的文件列表：

total 343700

-rwxr-xr-x 1 root root 31316 2007-12-05 08:58 [

Move the cursor to the third line, then enter the following ex command:

移动光标到第三行，然后输入以下 ex 命令：

:r foo.txt

The :r command (short for “read”) inserts the specified file before the cursor position. Our screen should now look like this:

这个:r 命令（是”read”的简称）把指定的文件插入到光标位置之前。现在屏幕应该看起来像这样：

total 343700

-rwxr-xr-x 1 root root 31316 2007-12-05 08:58 [

....

The quick brown fox jumped over the lazy dog. It was cool.

Line 2

Line 3

Line 4

Line 5

-rwxr-xr-x 1 root root 111276 2008-01-31 13:36 a2p

....

Saving Our Work

**保存工作**

Like everything else in vi, there are several different ways to save our edited files. We have already covered the ex command :w, but there are some others we may also find helpful.

像 vi 中的其它操作一样，有几种不同的方法来保存我们所修改的文件。我们已经研究了:w 这个 ex 命令， 但还有几种方法，可能我们也觉得有帮助。

In command mode, typing ZZ will save the current file and exit vi. Likewise, the ex command :wq will combine the :w and :q commands into one that will both save the file and exit.

在命令模式下，输入 ZZ 就会保存并退出当前文件。同样地，ex 命令:wq 把:w 和:q 命令结合到 一起，来完成保存和退出任务。

The :w command may also specify an optional filename. This acts like “Save As…” For example, if we were editing foo.txt and wanted to save an alternate version called foo1.txt, we would enter the following:

这个:w 命令也可以指定可选的文件名。这个的作用就如”Save As…“。例如，如果我们 正在编辑 foo.txt 文件，想要保存一个副本，叫做 foo1.txt，那么我们可以执行以下命令：

:w foo1.txt

Note: While the command above saves the file under a new name, it does not change the name of the file you are editing. As you continue to edit, you will still be editing foo.txt, not foo1.txt.

注意：当上面的命令以一个新名字保存文件时，但它并没有更改你正在编辑的文件的名字。 如果你继续编辑的话，你还是在编辑文件 foo.txt，而不是 foo1.txt。